Alfio Quarteroni Fausto Saleri

Scientific Computing with MAT lAB

Alfio Quarteroni Ecole Poly technique Pederale de Lausanne 1015 Lausanne, Switzerland and Politecnico di Milano Via Bonardi 9 20133 Milano, Italy e-mail: alfio.quarteroni@epfl.ch Fausto Saleri Politecnico di Milano Via Bonardi 9 20133 Milano, Italy e-mail: fausto.saleri@mate.polimi.it

Cover figure by Marzio Sala

Cataloging-in-Publication Data applied for Bibliographic information published by Die Deutsche Bibliothek Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

Title of the Italian original edition: Introduzione al Calcolo Scientifico. Springer-Verlag Italia, Milano, 2001, ISBN 88-470-0149-8

Mathematics Subject Classification (2000): 65-01, 68U01, 68N15

ISSN 1611-0994 ISBN 3-540-44363-0 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law. Springer- Verlag Berlin Heidelberg New York a member of BertelsmannSpringer Science+ Business Media GmbH http://www.springer.de @ Springer-Verlag Berlin Heidelberg 2003 Printed in Germany The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use. Cover Design: Friedhelm Steinen-Broo, Estudio Calamar, Spain Cover production: design & production GmbH, Heidelberg Typeset by the authors. Printed on acid-free paper 46/3142/LK - 543 210

Introduction

"It is important to prove, but it is more important to improve."

This textbook is an introduction to Scientific Computing. We will illustrate several numerical methods for the computer solution of certain classes of mathematical problems that cannot be faced by paper and pencil. We will show how to compute the zeros or the integrals of continuous functions, solve linear systems, approximate functions by polynomials and construct accurate approximations for the

solution of differential equations. With this aim, in Chapter 1 we will illustrate the rules of the game that computers adopt when storing and operating with real and complex numbers, vectors and matrices. In order to make our presentation concrete and appealing we will adopt the programming environment MATLAB @ 1 as a faithful companion. We will gradually discover its principal commands, statements and con- structs. We will show how to execute all the algorithms that we intro- duce throughout the book. This will enable us to furnish an immediate quantitative assessment of their theoretical properties such as stability, accuracy and complexity. We will solve several problems that will be raised through exercises and examples, often stemming from specific ap- plications.

1 MATLAB is a trademark of TheMath Works Inc., 24 Prime Park Way, N atick, MA 01760, Tel: 001+508-647-7000, Fax: 001+508-647-7001.

Several graphical devices will be adopted in order to render the read- ing more pleasant. We will report in the margin the MATLAB command along side the line where that command is being introduced for the first time. The symbol

will be used to indicate the presence of exercises, the symbol

to indicate the presence of a MATLAB program, while ,\

the symbol will be used when we want to attract the attention of the reader on a critical or surprising behavior of an algorithm or a pro- cedure. The mathematical formulae of special relevance are put within a frame. Finally, the symbol

indicates the presence of a display panel summarizing concepts and conclusions which have just been reported and drawn. At the end of each chapter a specific section is devoted to mentioning those subjects which have not been addressed and indicate the bibliographical references for a more comprehensive treatment of the material that we have carried out. Quite often we will refer to the textbook [QSSOO] where many issues faced in this book are treated at a deeper level, and where theoretical results are proven. For a more thorough description of MATLAB we refer to [HHOO]. All the programs introduced in this text can be downloaded from the web address

mox.polimi.it/Springer.

No special prerequisite is demanded of the reader, with the exception of an elementary course of Calculus. However, in the course of the first chapter, we recall the principal re- sults of Calculus and Geometry that will be used extensively throughout this text. The less elementary subjects, those which are not so neces- sary for an introductory educational path, are highlighted by the special symbol'll.

We express our thanks to Thanh-Ha Le Thi from Springer-Verlag Hei- delberg, and to Francesca Bonadei and Marina Forlizzi from Springer- Italia for their friendly collaboration throughout this project. We grate- fully thank Prof. Eastham of Cardiff University for editing the language of the whole manuscript and stimulating us to clarify many points of our text.

Milano and Lausanne, May 2003

Alfio Quarteroni, Fausto Saleri

Contents

1. What can't be ignored 1 1.1 Real numbers . . . . . .
. . . .. 1 1.1.1 How do we represent them . . . . 2 1.1.2 How do we operate with floating-point numbers. 4 1.2 Complex numbers . .. . 6 1.3 Matrices..... ..... 8 1.3.1 Vectors. .... 13 1.4 Real functions. ... . 14 1.4.1 The zeros. . . . . 16 1.4.2 Polynomials ... . 17 1.4.3 Integration and differentiation . . . . 19 1.5 To err is not only human . . . . 21 1.5.1 Talking about costs . . . . . 24 1.6 A few more words about MATLAB . .. ... 26 1.6.1 MATLAB statements . . . . 29 1.6.2 Programming in MATLAB . . . . . 30 1.7 What we haven't told you . . . . . 34 1.8 Exercises..... ... 34

2. Nonlinear equations 2.1 The bisection method . . 2.2 The Newton method ..

37 38 42

2.3 Fixed point iterations . . . . . 47 2.3.1 How to terminate fixed point iterations . . . . . 50 2.4 What we haven't told

you . . . . . 51 2.5 Exercises 53
3. ApproxiDlation of functions and data 57 3.1 Interpolation . . . . . 59 3.1.1 Lagrangian polynomial interpolation . . . . . 60 3.1.2 Chebyshev interpolation. . . . . 66 3.1.3 Trigonometric interpolation and FFT . . . . . 67 3.2 Piecewise linear interpolation . . . . . 72 3.3 Approximation by spline functions. . . . . 74 3.4 The least squares method . . . . 76 3.5 What we haven't told you . . . . . 80 3.6 Exercises 81

4. N uDlerical differentiation and integration 83 4.1 Approximation of function derivatives. . . . . 84 4.2 Numerical integration . .. . 87 4.2.1 Midpoint formula . . . . 87 4.2.2 Trapezoidal formula ... . 89 4.2.3 Simpson formula .. .. 92 4.3 Simpson adaptive formula . . . . .. 95 4.4 What we haven't told you . . . . 99 4.5 Exercises..... 100

5. Linear systeDls 103 5.1 The L U factorization method . . . . . 106 5.2 The technique of pivoting. . . . . 113 5.3 How accurate is the L U factorization? . . . . . 115 5.4 How to solve a tridiagonal system . . . . . 120 5.5 Iterative methods . . . . . 122 5.5.1 How to construct an iterative method . . . 123 5.6 When should an iterative method be stopped? . . 128 5.7 Richardson method . . . . . 130 5.8 What we haven't told you . . . . . 134 5.9 Exercises 134

6. Eigenvalues and eigenvectors 6.1 The power method .. . . . . 6.1.1 Convergence analysis ..... 6.2 Generalization of the power method. . 6.3 How to compute the shift. . . . . .

137 . 140 . 142 . 144 . 146

Contents

6.4 6.5 6.6

Computation of all the eigenvalues ... What we haven't told you Exercises . . . . . .

7. Ordinary differential equations 7.1 The Cauchy problem . . . . . 7.2 Euler methods . . . . . 7.2.1 Convergence analysis . . . . . 7.3 The Crank-Nicolson method . . . . . 7.4 Zero-stability . . . . . 7.5 Stability on unbounded intervals . . . . 7.5.1 Absolute stability controls perturbations . . . 7.6 High order methods . . . . . 7.7 The predictor-corrector methods . . . . 7.8 Systems of differential equations . . . . . 7.9 What we haven't told you . . . 7.10 Exercises . . . . . .

8. Numerical methods for boundary-value problems 8.1 Approximation of boundary-value problems . . . 8.1.1 Approximation by finite differences .... 8.1.2 Approximation by finite elements . . . 8.2 Finite differences in 2 dimensions . . . . . 8.2.1 Consistency and convergence. ... 8.3 What we haven't told you . . . . . 8.4 Exercises..... . . .

9. Solutions of the exercises 9.1 Chapter 1 . . . . . 9.2 Chapter 2 . . . . . 9.3 Chapter 3 . . . . . 9.4 Chapter 4 . . . . . 9.5 Chapter 5 . . . . . 9.6 Chapter 6 . . . . . 9.7 Chapter 7 . . . . . 9.8 Chapter 8 . . . . . .

Bibliography

Index of MATLAB Programs

Index

. IX

148 . . 149 150

153 . . 155 156 158 . . 162 163 . . 165 168 174 176 . 179 . . 182 184

187 . . 190 190 192 195 202 204 . . 205

207 . . 207 . . 209 . . 215 . . 219 . . 223 . . 228 . . 232 . . 241

245

249

1. What can't be ignored

In this book we will systematically use elementary mathematical concepts which the reader should know already, yet he or she might not recall them immediately. We will therefore use this chapter to refresh them, and to introduce new concepts as well which pertain to the field of Numerical Analysis. We will begin to explore their meaning and usefulness with the help of MAT- LAB (MATrix LABoratory), an integrated environment for programming and visualization in scientific computing. In Section 1.6 we will give a quick introduction to MATLAB, which is sufficient for the use that we are going to make in this book. However, we refer the interested readers to the manual [HHOO] for a complete description of this language. In the present Chapter we have therefore condensed notions which are typical of courses in Calculus, Linear Algebra and Geometry, yet rephrasing them in a way that is suitable for use in scientific computing.

While the set JR of real numbers is known to everyone, the way in which computers treat them is perhaps less well known. On one hand, since machines have limited resources, only a subset JF of finite dimension of IR can be represented. These are called floating-point numbers. On the other hand, as we shall see in Section 1.1.2, JF is characterized by properties that are different from those of JR . The reason is that any real number x is in principle truncated by the machine, giving rise to a new number (called the floating point number), denoted by fl(x), which does not necessarily coincide with the original number x.

2 1. What can't be ignored

To become acquainted with the differences between]R and JF, let us make a few experiments using MATLAB which illustrate the way that a com- puter (e.g. a PC) deals with real numbers. Whether we use MATLAB rather than another language is just a matter of convenience. The re- sults of our calculation, indeed, depend primarily on the manner in which the computer works, and only to a lesser degree on the programming language. Let us co nsider the rational number x == 1/7, whose decimal representation is 0.142857. This is an infinite representation, since the number of decimal digits is infinite. To get its computer representation, let us introduce after the prompt the ratio 1/7 and obtain > > 1/7 ans == 0.1429

which is a number with only four decimal digits, the last being different from the fourth digit of the original number. Should we now consider 1/3 we would find 0.3333, so the fourth decimal digit would now be exact. This behavior is due to the fact that real numbers are rounded on the computer. This means, first of all, that only a fixed number of decimal digits are returned, and moreover the last decimal digit which appears is increased by unity whenever the first disregarded decimal digit is greater than or equal to 5. The first remark to make is that using only four decimal digits to rep- resent real numbers it is questionable. Indeed, the internal representation of the number is made with as many as 16 decimal digits, and what we have seen is simply one of several possible MATLAB output formats. The same number can take different expressions depending upon the specific format format declaration that is made. For instance, for the number 1/7, some possible output formats are:

format long format short e format long e format short g format long g

yields "

"

0.14285714285714, 1.4286e - 01, 1.428571428571428e - 01, 0.14286, 0.142857142857143.

"

"

Some of them are more coherent than others with the internal computer representation. As a matter of fact, in general a computer stores a real number in the following way

x == (-1) 8 · (0. a 1 a2 . . . at) · (3e == (- 1) 8 · m · (3e - t , a 1 =I- 0, (1.1)

where s is either 0 or 1, {3 (a positive integer larger than or equal to 2) is the basis adopted by the specific computer at hand, m is an integer called the mantissa whose length t is the maximum number of digits ai (ranging between 0 and (3 -1) that are stored, and e is an integral number called the exponent. The format long e is the one which most resembles this representation, and e stands for exponent; its digits, preceded by the sign, are reported to the right of the character e. The numbers whose form is given in (1.1) are called floating-point numbers, since the position of the decimal point is not fixed. The digits a1 a2 . . . a p (with p < t) are often called the first p significant digits of x. The condition a1 i= 0 ensures that a number cannot have multiple representations. For instance, without this restriction the number 1/10 could be represented (in the decimal basis) as 0.1 · 10 0 , but also as 0.01 · 10 1 , etc.. The set IF is therefore completely characterized by the basis {3, the number of significant digits t and the range (L, U) (with L < 0 and U > 0) of variation of the index e. Thus it is denoted as IF({3, t, L, U). For instance, in MATLAB we have IF == JF(2, 53, -1021,1024) (indeed, 53 significant digits in basis 2 correspond to the 15 significant digits that are shown from MATLAB in basis 10 with the format long). Fortunately, the roundoff error that is inevitably generated whenever a real number x i= 0 is replaced by its representative f l (x) in IF, is small, . SInce

Ix - fl(x)1 1 < -E M IxI - 2 '

(1.2)

where EM == {31-t provides the distance between 1 and its closest floating- point number different from 1. Note that EM depends on (3 and t. For instance, in MATLAB EM can be obtained through the command eps, eps and we obtain EM == 2- 52

2.22.10- 16 . Let us point out that in (1.2) we estimate the relative error on x, which is undoubtedly more meaningful than the absolute error Ix - fl(x) I. As a matter of fact, the latter doesn't account for the order of magnitude of x whereas the former does. Number 0 does not belong to IF, as in that case we would have a1 == 0 in (1.1): it is therefore handled separately. Moreover, Land U being finite, one cannot represent numbers whose absolute value is either arbitrarily large or arbitrarily small. Precisely, the smallest and the largest positive real numbers of JF are given respectively by

. - (3 L-1 - aU (I a-t) Xm'I,n - , x max - fJ - fJ ·

"ealmin "ealmax

In MATLAB these values can be obtained through the commands realmin and realmax, yielding Xmin == 2.225073858507201. 10- 308 x max == 1.7976931348623158. 10+ 308 .

A positive number smaller than Xmin produces a message of under- flow and is treated either as 0 or in a special way (see, e.g., [QSSOO], Chapter 2). A positive number greater than X max yields instead a mes- Inf sage of overflow and is stored in the variable Inf (which is the computer representation of +00). The elements in JF are more dense near Xmin, and less dense while approaching Xmax. As a matter of fact, the number in JF nearest to X max (to its

left) and the one nearest to Xmin (to its right) are, respectively x m ax == 1.7976931348623157 · 10+ 308 X

in == 2.2250738585072O2 · 10- 308 .

Thus X

in - Xmin rv 10- 323 , while X max - x m ax

10 292 (!). However, the relative distance is small in both cases, as we can infer from (1.2).

Since JF is a proper subset of JR, elementary algebraic operations on float- ing point numbers do not enjoy all the properties of analogous oper- ations on JR. Precisely, commutativity still holds for addition (that is fl(x + y) == fl(y + x)) as well as for multiplication (fl(xy) == fl(yx)), but other properties such as associativity and distributivity are violated. Moreover, 0 is no longer unique. Indeed, let us assign the variable a the value 1, and execute the following instructions: > > a == 1; b==l; while a+b - == a; b==b/2; end The variable b is halved at every step as long as the sum of a and b remains different (-=) from a. Should we operate on real numbers, this program would never end, whereas in our case it ends after a finite number of steps and returns the following value for b: 1.1102e-16== EM /2. There exists therefore at least one number b different from 0 such that a+b=a. This is possible since IF is made up of isolated numbers; when adding two numbers a and b with b<a and b less than EM, we always obtain that a+b is equal to a. Associativity is violated whenever a situation of overflow or underflow occurs. Take for instance a=l. Oe+308, b=l. le+808 and c=-l. 001e+308, and carry out the sum in two different ways. We find that a + (b + c) == 1.0990e + 308, (a + b) + c == Inf.

This is a particular instance of what occurs when one adds two numbers with opposite sign but similar absolute value. In this case the result may be quite inexact and the situation is referred to as loss, or cancel- lation, of significant digits. For instance, let us compute ((1 + x) - l)/x (the obvious result being 1 for any x -# 0):

e""

» x == 1.e - 15; ((1 + x) - l)/x ans == 1.1102

This result is rather imprecise, the relative error being larger than 11 %1. Another case of numerical cancellation is encountered while evaluating the function

f(x) == x 7 - 7x 6 + 21x 5 - 35x 4 + 35x 3 - 21x2 + 7x - 1 (1.3) at 401 equispaced points with abscissa in [1 - 2 · 10- 8 ,1 + 2 · 10- 8]. We obtain the chaotic graph reported in Figure 1.1 (the real behavior is that of (x - 1)7, which is substantially constant and equal to the null function in such a tiny neighborhood of x == 1). In Section 1.4 we will see the commands that have generated the graph.

10- 14 1.S X

1 ' , , , , , , , , , :' , , , , , , , , , , , ': . . . ,. , . , , , , :' , , . , . , , , , , , ,

' , , , , , , . , , , :. , . - - . . . - . - . - - - - . . - - - - - - .

. - ......-.... """" ...., ..

-1 """."..:.....,..."., ':..",. ,..,:'",.,.,',.,

""""""""" """""""" ':' ,.,..,"" ':""" """

Fig. 1.1. Oscillatory behavior of the function (1.3) caused by cancellation errors

Finally, it is interesting to notice that in IF there is no place for indeterminate forms such as % or CXJ/CXJ, whose presence produces what is called not a number (NaN in MATLAB) for which the normal rules of NaN calculus do not apply.

Remark 1.1 Whereas it is true that roundoff errors are usually small, when repeated within long and complex algorithms, they may give rise to catas- trophic effects. Two outstanding cases concern the explosion of the Arianne missile on June 4, 1996, engendered by an overflow in the computer on board,

and the falling on an American barrack of an American Patriot missile, during the Gulf War in 1991, because of a roundoff error in the computation of its trajectory. An example with less catastrophic (but still troublesome) consequences is provided by the sequence Z2 == 2, Zn+l == 2n-l/2 J l - V l - 41-nz

, n == 2,3,... (1.4) which converges to 7r when n tends to infinity. When MATLAB is used to compute Zn, the relative error found between 7r and Zn decreases for the first 16 iterations, then grows because of roundoff errors (as shown in Figure 1.2).

10°

10-4 .....

10- 2 .

10-6 ."..,..

10-8

10- 10

10

15 20 25 30

, _ '

"'P ..:...:.:.;o:.....y...-.:...:.:.:.;....

Fig. 1.2. Logarithm of the relative error 17r - znl/7r versus n

See the Exercises 1.1-1.2.

complex

Complex numbers, whose set is denoted by C, have the form z == x + iy where i == F1 is the imaginary unit (that is i 2 == -1), while x == Re(z) and y == Im(z) are the real and imaginary part of z, respectively. They are generally represented on the computer as pairs of real numbers. Unless redefined otherwise, MATLAB variables i as well as j denote the imaginary unit. To introduce a complex number of real part x and imaginary part y, one can just write x+i*y; as an alternative, one can use the command complex (x, y). Let us also mention the trigonometric (or polar) representation of a complex number z,

z == pe iO == p(cos () + i sin ()),

(1.5)

where p = vi x 2 + y2 is the absolute value of the complex number (it can abs be obtained by setting abs (z)) and () the argument, that is the angle between the vector z of components (x, y), and the x-axis. () can be found angle by typing angle (z). The representation (1.5) is therefore:

1.2 Complex numbers 7

abs(z) * (cos(angle(z)) + i * sin(angle(z))).

The polar representation of one or more complex numbers can be obtained through the command compass (z) where z is either a single compass complex number or a vector whose components are complex numbers. For instance, by typing > > z == 3+i*3; compass(z); one obtains the graph reported in Figure 1.3.

90

5

180 ..... .:".... .

....:.'.... :.' ....:y<..'.... '.:""" .:',..,. .:.....'.:'..... 0 --. -.. ....

.....

.:... .:.:". : :- "." .. '. ....'

4 ..... .

: 3

: 2 : '.

.. .' .. . . - .." '. ..' . .

::' ..... : :..... .....:. ,

" .

....' '. ....

." "..

.. ..'..

. , ...",. ,.." . .

'. .' .

270

Fig. 1.3. Output of the MATLAB command compass

For any given complex number z, one can extract its real part with the command real (z) and its imaginary part with imag (z). Finally, the real complex conjugate z == x - iy of z, can be obtained by simply writing imag conj(z). con] In MATLAB all operations are carried out by implicitly assuming that the operands as well as the result are complex. We may therefore find some apparently surprising results. For instance, if we compute the cube root of -5 with the MATLAB command (-5)'" (1/3), instead of -1.7099 . . . we obtain the complex number 0.8550 + 1.4809i. (We anticipate the use of the symbol ,.. for the power exponent.) As a matter of fact, all num- ,.. bers of the form pe i ((}+2k1f), with k an integer, are indistinguishable from z. By computing {IZ we find f!Pe i (O/3+2k1r/3), that is, the three distinct roots

Zl = f!Pe iO / 3 , Z2 = f!Pei(O/3+27r/3), Z3 = f!Pe i (O/3+47r/3). MATLAB will select the one that is encountered by spanning the

complex plane counterclockwise beginning from the real axis. Since the polar

8 1. What can't be ignored

representation of z == -5 is pe i8 with p == 5 and () == -7r, the three roots are (see Fig. 1.4 for their representation in the Gauss plane) Zl ==

(cos(-7r /3) + i sin(-7r /3))

0.8550 - 1.4809i, Z2 ==

(cos(7r /3) + i sin(7r /3))

0.8550 + 1.4809i, Z3 ==

(cos(-7r) +isin(-7r))

-1.71. The second root is the one which is selected.

Im(z)

Z3 .....

Re(z)

Fig. 1.4. Representation in the complex plane of the three cube roots of the real number -5

Finally, by (1.5) we obtain the Euler formula: cos(O) =

(e iB + e- iB), sin(O) = 2

(e iB - e- iB) . (1.6)

Let nand m be positive integers. A matrix with m rows and n columns is a set of m x n elements aij, with i == 1, . . . , m, j == 1, . . . , n, represented by the following table:

all al2 al n a21 a22 a2n A== (1.7) aml a m 2 a mn

In compact form we write A == (aij). Should the elements of A be real numbers, we write A E]Rmxn, and A E c mxn if they are complex.

1.3 Matrices 9

Square matrices of dimension n are those with m == n. A matrix featuring a single column is a column vector, whereas a matrix featuring a single row is a row vector. In order to introduce a matrix in MATLAB one has to write the el- ements from the first to the last row, introducing the character ; to separate the different rows. For instance, the command

> > A == [1 2 3; 4 5 6]

produces

A== 1 4

2 5

3 6

that is, a 2 x 3 matrix whose elements are indicated above. The null ma- trix 0 is that with null elements aij == 0 for i == 1, . . . , m, j == 1, . . . , n; it can be constructed by the MATLAB command zeros (m, n). The MAT - zeros LAB command eye (m, n) produces a rectangular matrix whose elements eye are all 0 except those on the main diagonal which are unity. The main diagonal of a m x n matrix A is the diagonal made of elements aii, i == 1,. . ., min(m, n). A particular case is the command eye (n) (which is a shorthand version for eye (n, n)); it produces a square matrix of dimension n which is called the identity matrix and is denoted by I. We can define the following operations:

1. if A == (aij) and B == (b ij) are m x n matrices, the sum of A and B is the matrix A + B == (aij + b ij); 2. the product of a matrix A by a real or complex number A is the matrix AA == (Aaij);

3. the product of two matrices is possible only for compatible sizes, precisely if A is m x p and B is p x n, for some positive integer p. In that case C == AB is an m x n matrix whose elements are

P Cij = Laikbkj, for i = 1, . . . , m, j = 1, . . . , n. k==l

Here is an example of the sum and product of two matrices.

» A==[1 2 3; 4 5 6]; B==[7 8 9; 10 1112]; (==[13 14; 15 16; 17 18]; » A+B ans == 8 10 12

10 1. What can't be ignored

14 16 18 » A*C ans == 94 100 229 244

Note that MATLAB returns a diagnostic message when one tries to carry out operations on matrices with incompatible dimensions. For instance:

> > A+C ??? Error using ====> + Matrix dimensions must agree. » A*B ??? Error using ====> * Inner matrix dimensions must agree.

size Remark 1.2 For any given matrix A, the MATLAB command dim=size (A) returns the two-element variable dim= [m , n] containing the number of rows whos and columns of A. More in general, the command whos allows one to check the type of all variables in use in a work session. For instance, if we have defined only the matrices A, Band C as before, the command whos returns

Name Size Bytes Class A 2x3 48 double array B 2x3 48 double array C 3x2 48 double array Grand total is 18 elements using 144 bytes

If A is a square matrix of dimension n, its inverse (provided it exists) is a square matrix of dimension n, denoted by A -1, which satisfies the matrix relation AA -1 == A -1 A == I. We can obtain A -1 through the inv command inv (A). The inverse of A exist iff the determinant of A, a number denoted by det(A), is non-zero. The latter condition is satisfied in turn iff the column vectors of A are linearly independent (see Section 1.3.1). The determinant of a square matrix is defined by the following recursion formula (Laplace rule):

det(A) ==

n L

ijaij, for n > 1, Vi = 1,... ,n, j==l

(1.8)

all

if n == 1,

where

ij = (-1)i+ j det(A ij) and A ij is the matrix obtained byelim- inating the i-th row and j-th column from matrix A. In particular, if

A E JRlxl we set det(A) == all; if A E ffi.2x2 one has

det(A) == all a22 - a12 a 21;

if A E JR 3x3 we obtain

det(A) == alla22 a 33 + a3l a l2 a 23 + a2lal3a32

-alla23 a 32 - a2l a l2 a 33 - a31 a 13 a 22.

For the matrix product we have the following property: if A == BC, then det(A) == det(B) · det(C). To invert a 2 x 2 matrix and compute its determinant we can proceed as follows:

> > A==[1 2; 3 4]; > > inv(A) ans ==

-2.0000 1.5000 > > det(A) ans == -2

1.0000 -0.5000

Should a matrix be singular, MATLAB returns a diagnostic message, followed by a matrix whose elements are all equal to Inf, as illustrated by the following example: » A==[1 2; 0 0]; > > inv(A) Warning: Matrix is singular to working precision. ans == I nf I nf Inf Inf

For special classes of square matrices, the computation of inverses and determinants is rather simple. In particular, if A is a diagonal matrix, i.e. one for which only the diagonal elements akk, k == 1, . . . , n, are non-zero, its determinant is given by det(A) == alla22 · · · ann. In particular, A is non-singular iff akk f 0 for all k. In such a case the inverse of A is still a diagonal matrix with elements a k 1 . Let v be a vector of dimension n. The MATLAB command diag(v) diag produces a diagonal matrix whose elements are the components of vec- tor v. The more general command diag(v,m) yields a square matrix of dimension n+abs (m) whose m-th upper diagonal (i.e. the diagonal made of elements of indices i, i + m) has elements equal to the components

of v, while the remaining elements are null. Note that this extension is valid also when m is negative, in which case the only affected elements are those of lower diagonals. For instance if v = [1 2 3J then:

» A==diag(v,-l) A== 0 0 0 0 1 0 0 0 0 2 0 0 0 0 3 0

Other special matrices are the upper triangular and lower triangular matrices. A square matrix of dimension n is lower (respectively, upper) triangular if all elements above (respectively, below) the main diagonal are zero. Its determinant is simply the product of the diagonal elements. tril Through the commands tril (A) and triu (A), one can extract from triu the matrix A of dimension n its lower and upper triangular part. Their extensions tril(A,m) or triu(A,m), with m ranging from -n and n, permit the extraction of the triangular part augmented by, or deprived of, m extradiagonals. For instance, given the matrix A = [3 1 2; -1 3 4; -2 -1 3J, by the command L1=tril (A) we obtain

Ll == 300 -1 3 0 -2 -1 3

while, by L2=tril (A, -1) , we obtain

L2 == 000 -1 0 0 -2 -1 0

Finally, we recall that if A E IRmxn its transpose AT E IRnxm is the matrix obtained by interchanging rows and columns of A. When A == AT the matrix A is called symmetric. The MATLAB notation for A ' the transpose of A is A' .

Vectors will be indicated in boldface; precisely, v will denote a column vector whose i-th component is denoted by Vi. When all components are real numbers we can write v E

n. In MATLAB, vectors are regarded as particular cases of matrices. To introduce a column vector one has to insert between square brackets the values of its components separated by semi-colons, whereas for a row vector it suffices to write the component values separated by blanks or commas. For instance, through the instructions v = [1; 2; 3] and w = [1 2 3] we initialize the column vector v and the row vector w, both of dimension 3. The command zeros (n, 1) (respectively, zeros (1 ,n)) zeros produces a column (respectively, row) vector of dimension n with null elements, which we will denote by o. Similarly, the command ones(n, 1) ones generates the column vector, denoted with 1, whose components are all equal to 1. Finally, with the command v= [ ] we initialize an empty vector. A system of vectors {YI, . . . , Y m} is called linearly independent if the relation

alYI + · · · + amYm == 0

implies that all coefficients al,..., am are null. A system B == {YI,..., Yn} of n linearly independent vectors in

n (or en) is a basis for

n (or en), that is, any vector w in

n can be written as

n W = LWkYk, k==1

for a unique possible choice of the coefficients {Wk}. The latter are called the components of w with respect to the basis B. For instance, the canon- ical basis of

n is the set of vectors {el, . . . , en}, where ei has its i-th component equal to 1, and all other components equal to o. Although not a unique basis of

n, the latter is the one which is normally used. The scalar product of two vectors v, w E

n is defined as

n (v, w) = w T V = LVkWk, k==1

{Vk} and {Wk} being the components of v and w, respectively. The corresponding MATLAB command is w' *v, where now the apex

denotes transposition of a vector. The length (or modulus) of a vector v is given

by

n IIvII = J (v, v) = LV

k==I

norm and can be computed through the command norm (V) · . * The MATLAB command x. *y or x. ....2 indicates that these operations A should be carried out component by component. For instance if we define the vectors

» v == [1; 2; 3]; w == [4; 5; 6];

the instruction » w'*v ans == 32

provides their scalar product, while > > w. *v ans == 4 10 18

returns a vector whose i-th component is equal to XiYi. Finally, we recall that a number A (real or complex) is an eigenvalue of the matrix A E IRnxn, if

.:. . - ..... . ::::::{::' . .)

:

::::::::::"" .. . m..... . ...-. .... .., .'..
.....,..... ....

Av == AV, for a suitable v E en, v 1:- 0, which is called an eigenvector associated with A. In general, the computation of eigenvalues is quite difficult. Exceptions are represented by diagonal and triangular matrices, whose eigenvalues are their diagonal elements.

See the Exercises 1.3-1.5.

fplot

This chapter will deal with real functions. In particular, given a function f which is defined on an interval (a, b), we would like to compute its zeros, its integral and its derivative, as well as to determine its behavior. The command fplot(fun,lims) plots the graph of the function fun (which is stored as a string of characters) on the interval (1 ims (1) ,1 ims (2)). For instance, to represent f(x) == 1/(1 + x2) on (-5,5), we can write

» fun ==’I/(I+x A 2)’; lims==[-5,5]; fplot(fun,lims);

or, more directly,

» fplot('I/(I+xA2)',[-5 5]);

The graph is obtained by sampling the function on a set of non-equispaced abscissae and reproduces the true graph of f with a tolerance of 0.2%. To improve the accuracy we could use the command > > fplot(fun, lims,tol ,n,' LineSpec' , PI, P2,...) where tol indicates the desired tolerance and the parameter n(> 1) ensures that the function will be plotted with a minimum of n + 1 points. , LineSpec' is a given line specification (for instance, , --, for a dashed line, ':' for a dotted line), while the parameters P 1 , P2, . .. can be passed directly to the function fun. To use default values for tol, n or , LineSpec' one can pass empty matrices ([ ]). To evaluate a function fun at a point x we write y=eval (fun), after eval having initialized x. The corresponding value is stored in y. Note that x, and correspondingly y, can be a vector. When using this command, the restriction is that the argument of the function fun must be x. When the argument of fun has a different name (this is often the case when this argument is generated at the interior of a program) the command eval would be replaced by feval (see Remark 1.4). Finally, we point out that if we write grid on after the command grid fplot, we can obtain the background-grid as that in Figure 1.1.

Remark 1.3 In general circumstances, it might be useful to allow any num- ber of arguments to a function. This is made possible by using the variable varargin, a cell array containing the optional arguments to the function. varargin varargin must be declared as the last input argument and collects all the inputs from that point onwards. For instance, the program (see Section 1.6.2) function L==mytril(varargin) L==tril(varargin{:});

collects all the inputs starting into the variable varargin. mytril uses the comma-separated list syntax varargin{:} to pass the parameters to the func- tion tril. The call,

L==mytril([1 2 3; 4 5 6; 7 8 9] ,-2) L== o o 7

o o o

o o o

results in varargin being a l-by-2 cell array containing the values [1 2 3; 3 4 5; 6 7 8] and -2.

We recall that a is said to be a zero of a real function f if f(a) == o. It is simple if f' (a) i=- 0, multiple otherwise. From the graph of a function one can infer (within a certain toler- ance) which are its real zeros. The direct computation of all zeros of a given function is not always straightforward. For functions which are polynomials with real coefficients of degree n, that is, of the form

n Pn(x) = ao + alX + a2 x2 + . . . + anx n = Lak xk , ak E R, an i=- 0, k==O

we can obtain the only zero a == -ao/ aI, when n == 1 (i.e. PI represents a straight line), or the two zeros, a+ and a_, when n == 2 (this time P2 represents a parabola)

-al ::I::: V ai - 4aOa2 a::t: == 2a2

However, there are no explicit formulae for the zeros of an arbitrary polynomial Pn when n > 5. Also the number of zeros of a function cannot in general be determined a priori. An exception is provided by polynomials, for which the number of zeros (real or complex) coincides with the polynomial degree. More- over, should Q == X + iy be a zero of a polynomial, its complex conjugate Q == x - iy is also a zero. To compute in MATLAB one zero of a function fun, near a given fzero value xO, either real or complex, the command fzero(fun,xO) can be used. The result is an approximate value of the desired zero, and also the interval in which the search was made. Alternatively, using the command fzero (fun, [xO xl]), a zero of fun is searched for in the interval whose extremes are xO , xl, provided f changes sign between xO and xl. Let us consider, for instance, the function f (x) == x 2 - 1 + eX. Looking at its graph we see that there

are two zeros in (-1, 1). To compute them we need to execute the following commands:

» fun=='x A 2 - 1 + exp(x)'; » fzero(fun,l) Zero found in the interval: [-0.28, 1.9051]. ans ==

6.0953e-18 » fzero(fun,-l)

Zero found in the interval: [-1.2263, -0.68].

ans == -0.7146

In Chapter 2 we will introduce and investigate several methods for the approximate computation of the zeros of an arbitrary function.

Polynomials are very special functions and there is a special MATLAB toolbox, polyfun, for their treatment. The command polyval, is apt to polyval evaluate a polynomial at one or several points. Its input arguments are a vector p and a vector x, where the components of p are the polynomial coefficients stored in decreasing order, from an down to ao, and the components of x are the abscissae where the polynomial needs to be evaluated. The result can be stored in a vector y by writing

> > y == polyval(p,x) For instance, the values ofp(x) == x 7 +3x 2 -1, at the equispaced abscissae Xk == -1 + k * 0.25 for k == 0, . . . ,8, can be obtained by proceeding as follows:

» p == [1 0 000 3 0 -1]; x == [-1:0.25:1]; > > y == polyval(p,x) y== Columns 1 through 7 1.0000 0.5540 -0.2578 -0.8126 -1.0000 -0.8124 -0.2422 Columns 8 through 9 0.8210 3.0000

Alternatively, one could use the command fplot. However, in such case one should provide the entire analytic expression of the polynomial in the input string, and not simply its coefficients. Let us recall that if Q is such that p(Q) == 0, then Q is called zero of p or, equivalently, a root of the algebraic equation p(x) == o. The program roots provides an approximation of the zeros of a polynomial roots and requires only the input of the vector p. For instance, we can compute the zeros of p(x) == x 3 - 6x 2 + llx - 6 by writing

> > p == [1 -6 11 -6]; format long; > > roots(p) ans == 3.00000000000000 2.00000000000000 1.00000000000000

-

Unfortunately, the result is not always that accurate. For instance, for the polynomial p(x) (x - 1)7, whose unique zero is Q == 1, we find (quite surprisingly)

» p == [1 -7 21 -35 35 -21 7 -1]; > > roots(p) ans == 1.0088 1.0055 + 0.0069i 1.0055 - 0.0069i 0.9980 + 0.0085i 0.9980 - 0.0085i 0.9921 + 0.0038i 0.9921 - 0.0038i

This inaccuracy is due to the fact that the coefficients of p have alternating signs, yielding severe cancellation errors. conv We mention that using the command p=convCpi ,p2) we can obtain the coefficients of the polynomial given by the product of two polynomials whose coefficients are contained in the vectors pi and p2. Simi- deconv larly, the command [q, r] =deconv Cpi , p2) provides the coefficients of the polynomials obtained on dividing pi by p2, i.e. pi = convCp2,q) + r. In other words, q and r are the quotient and the reminder of the division. Let us consider for instance the product and the ratio between the two polynomials PI (x) == x 4 - 1 and P2 (x) == x 3 - 1 :

»pl == [1 000 -1]; > > p2 == [1 0 0 -1]; > > p==conv(pl,p2) p == 1 0 0 -1 -1 > > [q,r]==deconv(p1,p2) q== 1 0

o

o

1

r==

o

o

o

1 -1

We therefore find the polynomials p(x) == PI (x)P2 (x) == X 7 - x 4 - x3 + 1, q(x) == x and r(x) == x-I such that PI (x) == q(X)p2(X) + r(x). Finally, the commands polyint(p) and polyder(p) provides respec- polyint tively the coefficients of the primitive (vanishing at x == 0) and those polyder of the derivative of the polynomial whose coefficients are given by the components of the vector p. If x is a vector of abscissae and p (respectively, Pi) is a vector con- taining the coefficients of a polynomial P (respectively, Pi), the previous commands are summarized in the following table:

command yields y=polyval(p,x) y == values of P(x) z=roots(p) z == roots of P such that P(z) == 0 p=conv (PI ,P2) p == coefficients of the polynomial PI P 2 [q, r] =deconv (PI ,P2) q == coefficients of Q, r == coefficients of R such that PI == Q P2 + R y=polyder(p) y == coefficients of pi (x) y=polyint(p) y == coefficients of J P(x) dx

A further command, polyfi t, allows the computation of the n + 1 polynomial coefficients of a polynomial P of degree n once the values attained by P at n + 1 distinct nodes are available (see Section 3.1.1).

The following two results will often be invoked throughout this book.
1. the fundamental theorem of integration: if f is a continuous function in [a, b), then

x F(x) = j f(t) dt

a

is a differentiable function, called a primitive of f, which satisfies, Vx E [a, b),

pi (x) == f (x) ;

2. the first mean-value theorem for integrals: if f is a continuous function in [a, b) and Xl, x2 E [a, b), then :3

E (Xl, X2) such that

X2 f(

) = 1 j f(t) dt. X2 - Xl

Xl

20 1. \Vhat can't be ignored

Even when it does exist, a prilllitive lllight be eitiler impossible t() determine or difficult to cOlrlpllte. For installce, knowing tilat In l

rl is a, primitive of l/x is irrelevant if one doesn't know how t() cOlnpute efficielltly the logarithms. III chapter 4 we will illtr()dIICe severa.! methods to compute the integral of arbitrary contilluol1S fllnctions with a desired accuracy, irrespectively of tlle kllowledge of its primitive. We reca.!l tllat a fl1llCtioll f defillecl on an interval [a, b] is differentia,ble in a I)oint x E (a, b) if tlle following limit exists a,nd is finite

1 .f'(x) == li111- } (f(x + IL) - .f(x)). h ---+0 11

(1.9)

If x == a tilis definitioll requires h to be positive, whereas 11 IllUSt be negative if x == b. In all cases, the value of tIle derivative provides the slope of the tangent line to the graph of f at the POillt X. We say tilat a functioll which is continuous together with its derivative at allY point of [a, b] belongs to the space C 1 ([a, b]). 1\/lore gellerally, a function witil continuous derivatives up to the order p (a positive integer) is said to belollg to CP([a, b]). In particular, CO ([a, b]) denotes the space of contin- uous functioI1S in [a, b]. A result that will be often used is the mearL value theorem, according to whicll, if f E C 1 ([a,b]), there exists

E (a, b) such that

f'(

) == (f(b) - f(a))/(b - a).

Finally, it is worth recalling that a function continuous with all its derivatives up to tIle order n + 1 in a neighborhood of Xo, can be ap- proximated in stIch a lleigllborhood by the so-called Taylor polynorrLial of degree n at the point Xo:

Tn(x) = J(xo) + (x - xo)J'(;:r;o) + ... +

(x - xotj(n)(xo) fl; . =

(x - xo)k f (k) (

) L-t k!

IO . k==O

syms

The MATLAB toolbox symbolic provides the COIIIIIIands diff, int and taylor wIlich allow tIS to obtaill the a,nalytical expression of tIle derivative, the indefinite integral (i.e. a, prilnitive) a.rld tIle Taylor poly- nomial, respectively, of a given function. In particlilar, Ilaving defined in the string f the function on which we intelld to operate, diff (f , n) provides its derivative of order n, int (f) its illdefinite integral, alld taylor (f , x, n+l) the associated Taylor polynolllia J of degree n in a neighborhood of Xo == O. The variable x 111ust be declared symbolic by using the command syms x. This will allow its algebraic manipulation

diff int taylor

1.5 To err is not ()nly human 21

without specifying its value. In order to do this for the fllnction f(x) == (x 2 + 2x + 2)/(x 2 - 1), we proceed as follows:

» f == '(x A 2+2*x+2)/(x A 2-1)'; » syms x > > diff(f) (2*x+2) / (x A 2-1)-2* (x A 2+2*x+2) / (x A 2-1) A2*x » int(f) x+5 /2*log(x-1)-1 /2*log(1 +x) > > taylor(f,x,6) -2-2*x-3*x A 2-2*x A 3-3*x A 4-2*x A 5

The comrrland funtool, by tIle graI)llical interfa,ce illustrated irl Fig. funtool 1.5, allows a very easy syrrlbolic marlipulation of arbitraTY functi()ns.

Figure No. 3

. . .. " ....-

Fig. 1.5. GraI)llical interface ()f the comrnand funtool

See the Exercises 1.6-1. 7.

.:... x,".. .:. """. """0' ::,")i .

As a matter of fact, by re-phrasing the Latin motto Errare humanum est, we might say that in nl1merical cornputation to err is even inevitable.

As we have seen, the simple fact of using a computer to represent real numbers introduces errors. What is therefore important is not to strive to eliminate errors, but rather to be able to control their effect.

Generally speaking, we can identify several levels of errors that occur during the approximation and resolution of a physical problem (see Figure 1.6).

MP

Xph II PP em , .-... T X X = ! <jJ(t)dt e c . - - 0 H et e a X n = L<jJ(tk)O:k NP - - k

Fig. 1.6. Types of errors in a computational process

At the highest level stands the error em which occurs when forcing the physical reality (P P stands for physical problem and Xph denotes its solution) to obey some mathematical model (M P, whose solution is x). Such errors will limit the applicability of the mathematical model to certain situations and are beyond the control of Scientific Computing. The mathematical model (whether expressed by an integral as in the example of Figure 1.6, an algebraic or differential equation, a linear or non linear system) is generally not solvable in explicit form. Its reso- lution by computer algorithms will surely involve the introduction and propagation of roundoff errors at least. Let's call these errors ea. On the other hand, it is often necessary to introduce further errors since any procedure of the mathematical model involving an infinite sequence of arithmetic operations cannot be performed by the computer unless approximately. For instance the computation of the sum of a series will necessarily be accomplished in an approximate way by considering a suitable truncation.

It will therefore be necessary to introduce a numerical problem, N P, whose solution X n differs from x by an error et which is called trunca- tion error. Such errors do not occur only in mathematical models that are already set in finite dimension (for instance, when solving a linear system). The sum of the errors e a and et constitutes the computational error e c , the quantity we are interested in. The absolute computational error is the difference between x, the exact solution of the mathematical model, and X, the solution obtained at the end of the numerical process,

e

bs == lx - xl,

while (if x

0) the relative computational error is

e

el == lx - xi/lxi,

where l · l denotes the modulus, or other measure of size, depending on the meaning of x. The numerical process is generally an approximation of the mathemat- ical model obtained as a function of a discretization parameter, which we will refer to as h and suppose positive. If, as h tends to 0, the nu- merical process returns the

solution of the mathematical model, we will say that the numerical process is convergent. Moreover, if the (absolute or relative) error can be expressed as a function of h as

e c == Ch P ,

(1.10)

where C is independent of hand p is a positive number, we will say that the method is convergent of order p.

Example 1.1 Suppose we approximate the derivative of a function I at a point x with the incremental ratio that appears in (1.9). Obviously, if I is differentiable at x, the error committed by replacing I' by the incremental ratio tends to 0 as h

O. However, as we will see in Section 4.1, the error can be considered as Ch only if I E C 2 in a neighborhood of x.

While studying the convergence properties of a numerical procedure we will often deal with graphs reporting the error as a function of h in a logarithmic scale, which shows log(h) on the abscissae axis and log(e c) on the ordinates axis. The purpose of this representation is easy to see: if e c == Ch P then log e c == log C + p log h. p in logarithmic scale therefore represents the slope of the straight line log e c , so if we must compare two methods, the one presenting the greater slope will be the one with a higher order. In MATLAB it is very simple to obtain graphs in a logarithmic scale: one just needs to type loglog (x, y), x and y being loglog

the vectors containing the abscissae and the ordinates of the data to be represented. For instance, in Figure 1.7 are reported the straight lines relative to the behavior of the errors in two different methods. The continuous line represents a first-order approximation, while the hatched line represents a second-order one.

4 10 I 2 I 10 I I 0 I 10 I I -2 I 10 I I I -4 I 10 I I -6 I 10 1 I I -8 I 10 I I -10 I 10 I I -12 I 10 I 2 I -14 I 10 I I -16 I 1 10 I -12 -10 -8 -6 -4 -2 0 10 10 10 10 10 10 10 10

Fig. 1.7. Plot in logarithmic scales

There is an alternative to the graphical way of establishing the order of a method when one knows the errors ei relative to some given values hi of the parameter of discretization, with i == 1,..., N: it consists in supposing that ei is equal to Chf where C does not depend on i. One can then approach P with the values:

Pi == log(ei/ei-l)/log(hi/h i - 1) i == 2,... ,N.

(1.11)

Actually the error is not a computable quantity since it depends on the unknown solution. Therefore it is necessary to introduce computable quantities that can be used to estimate the error itself, the so called error estimator. We will see some examples in Sections 2.2, 2.3 and 4.3.

In general a problem is solved on the computer by an algorithm, which is a precise directive in the form of a finite text specifying the execution of a finite series of elementary operations. We are interested in those algorithms which involve only a finite number of steps.

The computational cost of an algorithm is the number of floating-point operations (in short, ops) that are required for its execution. Often, the speed of a computer is measured by the maximum number of floating-point operations which the computer can execute in one second (flops). In particular, the following abridged notations are commonly used: Mega-flops, equal to 10 6 flops, Giga-flops equal to 10 9 flops, Tera- flops equal to 10 12 flops. The fastest computers nowadays reach as many as 40 of Tera-flops. Earlier versions of MATLAB allowed the count of the number of floating point operations by means of the command flops. This is no longer possible in MATLAB 6. However, through this book we will sometimes make use of the command flops; in those cases it is understood that the version 5.3 of MATLAB is being used. In general, the exact knowledge of the number of operations requested by a given algorithm is not essential. Rather, it is useful to determine its order of magnitude as a function of a parameter d which is related to the problem dimension. We therefore say that an algorithm has constant complexity if it requires a number of operations independent of d, i.

e. O(1) operations, linear complexity if it requires O(d) operations, or, more generally, polynomial complexity if it requires O(d m) operations, for a positive integer m. Other algorithms may have exponential (O(cd) operations) or even factorial (O(d!) operations) complexity. We recall that the symbol O(d m) means "it behaves, for large d, like a constant times d m " .

Example 1.2 (matrix-vector product) Le A be a square matrix of order n and let v E }Rn: the j - th component of the product Av is given by

aj1 V 1 + aj2 V 2 +... + ajnVn,

and requires n products and n - 1 additions. One needs therefore n(2n - 1) ops to compute all the components. Thus this algorithm requires O(n 2) ops, so it has a quadratic complexity with respect to the parameter n. The same algorithm would require O(n 3) ops to compute the product of two matrices of order n. However, there is an algorithm, due to Strassen, which requires "only" O(nlog27) ops, and another, due to Winograd and Coppersmith, requiring O(n 2 . 376) ops.

Example 1.3 (computation of a matrix determinant) As already mentioned, the determinant of a square matrix of order n can be computed by the recursive formula (1.8). The corresponding algorithm has a factorial complex- ity with respect to n and would be usable only for matrices of small dimen- sion. For instance, if n == 24, a computer capable of performing as many as 1 Peta-flops (i.e. 10 15 ops per second) would require 20 years to carry out this computation. One has therefore to resort to more efficient algorithms. In- deed, there exists an algorithm that allows the

computation of determinants through matrix-matrix products, with henceforth a complexity of O(n1og2 7) ops by resorting to the Strassen algorithm previously mentioned (see [BB96]).

The number of operations is not the sole parameter which matters in the analysis of an algorithm. Another relevant factor is represented by the time that is needed to access the computer memory (which de- pends on the way the algorithm has been coded). An indicator of the performance of an algorithm is therefore the CPU time (CPU stands for central processing unit), and can be obtained using the MATLAB com- cputime mand cputime. The total elapsed time between the input and output etime phases can be obtained by the command etime.

Example 1.4 In order to compute the time needed for a matrix-vector mul- tiplication we set up the following program:

```
> > A == ra nd (n, n); v == ra nd (n); T == [ ]; sizeA == [ ]; cou nt == 1; > > for k == l:step:n AA == A ( 1 : k, 1 : k); vv == v ( 1 : k) , ; t == cputime; b == AA*vv; tt == cputime - t; T == [T, tt]; sizeA == [sizeA,k]; count == count + 1; end
```

a: step: b The instruction a: step: b appearing in the for cycle generates all numbers having the form a+step*k where k is an integer ranging from 0 to the largest value kmax for which a+step*kmax is not greater than b (in the case at hand, rand a=1,

b=2000 and step=50). The command rand(n,m) defines an nxm matrix of random entries. Finally, T is the vector whose components contain the time of CPU needed to carry out every single matrix-vector product, whereas cputime returns the CPU time in seconds that has been used by the MATLAB process since MATLAB started. The time necessary to execute a single program is therefore the difference between the actual CPU time and the one computed before the execution of the current program which is stored in the variable t. Figure 1.8, which is obtained by the command plot(sizeA,T, '0')), shows that the CPU time grows like the square of the matrix order n.

MATLAB is an integrated environment for scientific computing and vi- sualization. It is written in C language and is distributed by The Math- Works (see the website www.mathworks.com). The main program is contained in the subdirectory bin of the principal directory matlab. Once installed, the execution of MATLAB allows access » to a working environment characterized by the prompt ». For instance, by executing MATLAB on our personal computer .it shows

0.4

0.35

0.3 .....

, , . . .; . , . . , ...; . . . , . , , . .; . . . , . . , , .:. . . . . , .:. . . . .:. . . . , ...:. . . .

. . . . . o

0.25 ,.....',. ....

. . . .,...."..,. ,*."..... ,',..,.,... ,"" ..... ,-,....,.,. ,-",.,.,., ,-"....,., - - - - . - . .: 0

: : . . _ 0 . 0.2 ..... ,....,...

"..,....

..,.....

,..,.....:.....:... 0..':....'....:..''... . . . :: 0 . . 0.15 .".. .. ..:.....:... ....,.....

.. . 0" ..

.., ...:".....:..... 4;) : . . 0.1 .. ..:.... O. .. .:.. .. ..:.. .., ...:.. .. ..:.. .. .. o - .

. . - : 0: - : : : 0.05 "....

..,...,. 0..... .:...'..... .:..... .:..... .:...'..'.. .:....'.... - - - - -

0:

o

o . 0 200 400 600 800 1 000 1200 1400 1600 1800 2000

Fig. 1.8. Matrix-vector product: the CPU time (in seconds) versus the di- mension n of the matrix (on a PC at 433 Mhz)

<MATLAB> Copyright 1984-2001 The MathWorks, Inc. Version 6.1.0.450 Release 12.1 May 18 2001

To get started, select" MAT LAB Help" from the Help menu.

» After pressing the key enter (or else return), all that is written af- ter the prompt will be interpreted. 1 Precisely, MAT LAB will first check whether what is written corresponds either to variables which have al- ready been defined or to the name of one of the programs or commands defined in MATLAB. Should all those checks fail, MAT LAB returns an error warning. Otherwise, the command is executed and an output will possibly be displayed. In all cases, at the end the system returns the prompt to acknowledge that it is ready for a new command. To close a MATLAB session one should write the command qui t (or else exi t) and press the key enter. From now it will be understood that to execute a program or a command one has to press the key enter. Moreover, the terms program, function or command will be used in an equivalent man- ner. When our command coincides with one of the elementary structures characterizing MAT LAB (e.g. a number or a string of characters that

are put among apices) they are immediately returned in output in the default variable ans (abbreviation of answer). Here is an example:

IThus a MATLAB program does not necessarily have to be compiled as other languages do, e.g. Fortran or C, although a MATLAB compiler may be invoked by the command mee to allow a faster execution

quit exit

ans

clear

save

load

help

sin cos sqrt exp

+ - * / & | - > >= < <= ==

» 'home' ans == home

If now we write a different string (or number), ans will assume this new value. We can turn off the automatic display of the output by writing a semicolon after the string. Thus if we write' home' ; MATLAB will simply return the prompt (yet assigning the value ' home' to the variable ans). More generally, the comlnand = allows the assignment of a value (or a string of characters) to a given variable. For instance, to assign the string , Welcome to NYC' to the variable a we can write

» a=='Welcome to NYC';

As we can see, there is no need to declare the type of a variable, MAT- LAB will do it automatically and dynamically. For instance, should we write a=5, the variable a will now contain a number and no longer a string of characters. This flexibility is not cost-free. If we set a variable named qui t equal to the number 5 we are inhibiting the use of the MATLAB command qui t. We should therefore try to avoid using variables having the name of MATLAB commands. However, by the command clear fol- lowed by the name of a variable (e.g. qui t), it is possible to cancel this assignment and restore the original meaning of the cOIIlmand qui t. Using the command save followed by the name fname all existing workspace variables are

saved in the binary MATLAB file fname .mat. These data may be retrieved with the command load fname. mat. Omit- ting the filename causes save (or load) to use the default filenalne mat lab . mat. To save the variables vi, v2, ..., vn the synthax is: save fname vi v2 ... vn. By the command help one can see the whole family of commands and pre-defined variables, including the so-called toolboxes which are sets of specialized cOlllmands. Among them let us recall those which define the elementary functions such as sine (sin(a)), cosine (cos (a)), square root (sqrt (a)), exponential (exp (a)). There are special characters that cannot appear in the name of a variable or in a command, for instance the algebraic operators (+, -, * and /), the logical operators and (&), or (I), not (

), the relational operators greater than (», greater than or equal to (>=), less than «), less than or equal to «=), equal to (==). Finally, a name can never begin with a digit, a bracket or with any punctuation mark.

A special programming language, the MATLAB language, is also avail- able enabling the users to write new programs. Although its knowledge is not required for understanding how to use the several programs which we will introduce throughout this book, it may provide the reader with the capability of modifying thenl as well as producing new ones. The MATLAB language features standard statements, such as condi- tionals and loops. The if-elseif-else conditional has the following general form:

if cond(l) statement(1) elseif cond (2) statement(2)

else statement(n) end

where e ond (1) , e ond (2) , ... represent MA TLAB sets of instructions , with values 0 or 1 (false or true) and the entire construction allows the execu- tion of that statement corresponding to the condition taking value equal to 1. Should all conditions be false, the execution of statement (n) will take place. This means that if the value of eond (k) is zero, the control moves on. For instance, to compute the roots of a quadratic polynomial ax 2 + bx + c one can use the following instructions (the command disp(.) simply displays what is written between brackets):

»ifa-==O sq == sqrt (b * b - 4 * a * e); x(l) == 0.5 * (-b + sq)fa; x(2) == 0.5 * (-b - sq)fa; else if b-== 0 x(l) == -efb; elseif e -== 0 disp('Impossible equation'); else disp('The given equation is an identity') end

(1.12)

Note that MATLAB does not execute the entire construction until the statement end is typed. MATLAB allows two types of loops, a for-loop (comparable to a For- tran do-loop or a C for-loop) and a while-loop. A for-loop repeats the statements in the loop as the loop index takes on the values in a given row vector. For instance, to compute the first 6 terms of the Fibonacci sequence {fi == fi-l + fi-2} with fl == 0 and f2 == 1, one can use the following instructions:

> > f(1) == 0; f(2) == 1; > > for i == [3 4 5 6] f(i) == f(i-1) + f(i-2); end

Note that a semicolon can be used to separate several MATLAB instruc- tions typed on the same line. Also, note that we can replace the second instruction by the equivalent » for i = 3: 6. The while-loop repeats as long as the given cond is true. For instance, the following set of in- structions can be used as an alternative to the previous set:

» f(1) == 0; f(2) == 1; k == 3; » while k <== 6 f(k) == f(k-1) + f(k-2); k == k + 1; end

Other statements of perhaps less frequent use exist, such as switch, case, otherwise. The interested reader can have access to their meaning by the help command.

.

Let us now explain briefly how to write MATLAB programs. As previ- ously noticed, new programs may be added to IVfATLAB. A new program must be put in a file with a given name with extension m, which is called M-file. They must be located in one of the directories in which MATLAB automatically searches for m-files; their list can be obtained by the com- path mand path (see help path to learn how to add a directory to this list). The first directory scanned by MATLAB is the current working directory. It is important at this level to distinguish between scripts and func- tions. A script is simply a collection of MATLAB commands in an m-file and can be used interactively. For

instance, the set of instructions (1.12) can give rise to a script (which we could name equation) by copying it in the file equation. m. To launch it, one can simply write after the

MATLAB prompt » the instruction equation. We report two examples below:

> > a == 1; b == 1; c == 1; > > equation ans == -0.5000 + 0.8660i -0.5000 - 0.8660i

> > a == 0; b == 1; c == 1; > > equation ans == -1

Much more flexible than scripts are functions. A function is, in general, defined in an m-file (which we will generically call name .m) beginning with a line of the following form: fun ct ion [0 utI, . . . ,0 u t n] == n a me (i n 1 , . . . , i n m) where out!, . . . , outn are the output variables and in! , . . . , inm are the input variables. The following file, called det23. m, defines a new function called det23 which computes, according to the formulae given in Section 1.3, the determinant of a matrix whose dimension could be either 2 or 3:

fu nction det==det23(A) %DET23 computes the determinant of a square matrix % of dimension 2 or 3 [n,m]==size(A); if n====m if n====2 det == A(1,1)*A(2,2)-A(2,1)*A(1,2); elseif n ==== 3 det == A(

1,1) *det23(A[2 ,3], [2,3]))-A(1,2) *det23(A([2 ,3], [1,3]))+... A(1,3)*det23(A[2,3], [1,2]));

else disp(' Only 2x2 or 3x3 matrices '); end else disp(' Only square matrices '); end return

Note the use of the continuation characters ... meaning that the in- struction is continuing on the next line. The instruction A ([i , j J , [k, 1J) allows the construction of a 2 x 2 matrix whose elements are the elements

of the original matrix A lying at the intersections of the i-th and j-th rows with the k-th and I-th columns. When a function is invoked, MATLAB creates a local workspace. The commands in the function cannot refer to variables from the global (in- teractive) workspace unless they are passed as inputs. In particular, vari- ables used in a function are erased when the execution terminates, unless they are returned as output parameters. The symbol % is used to begin comments. Usually functions terminate when the end of the function is reached, return however a return statement can be used to force an early return (ac- cording to the fulfillment of a certain condition). For instance, in order to approximate the golden section number a == 1.6180339887. . ., which is the limit for k

00 of the quotient !k/ !k-l, by iterating until the differ- ence between two consecutive ratios is less than 10- 4 , we can construct the following function: fu nction [golden, k] ==fi bonacci f(l) == 0; f(2) == 1; goldenold == 0; kmax == 100; tol == 1.e-04; for k == 3:kmax f(k)

== f(k-1) + f(k-2); golden == f(k)/f(k-1); if abs(golden - goldenold) <== tol retu rn end goldenold == golden; end retu rn

Then, we can write

» [alpha,niter]==fibonacci alpha == 1.61805555555556 niter == 14

After 14 iterations the function has returned an approximate value which shares with a the first 5 significant digits. The number of input and output parameters of a MATLAB function can vary. For instance, we could modify the Fibonacci function as follows:

fu nction [golden, k] ==fi bonacci (tol, kmax) if nargin ==== 0 kmax == 100; tol == 1.e-04; % default values elseif nargin ==== 1

kmax == 100; % default value only for kmax end f(l) == 0; f(2) == 1; goldenold == 0; for k == 3:kmax f(k) == f(k-1) + f(k-2); golden == f(k)/f(k-1); if abs(golden - goldenold) <== tol retu rn end goldenold == golden; end retu rn

The function nargin counts the number of input parameters. In the nargln new version of the fUIlction fibonacci we can prescribe the maximum number of inner iterations allowed (kmax) and a specific

tolerance tal. When this information is missing the function must provide default val- ues (in our case, kmax = 100 and tal = 1. e-04). A possible use of it is as follows:

> > [a I pha, niter] ==fi bonacci (1.e-6, 200) alpha == 1.61803381340013 niter == 19

Note that using a stricter tolerance we have obtained a new approximate value that shares with Q as many as 8 significant digits. The function nargin can be used externally to a given function to obtain the nunlber of input parameters. Here is an example: » nargin('fibonacci') ans == 2

Remark 1.4 (inline functions) The command inline, whose most simple inline synthax reads g=inline (expr , arg1 , arg2, . . . ,argn), declares a function g which depends on the strings arg1, arg2, . . . ,argn. The string expr contains the expression of g. For instance, g=inline (, sin (r) , , 'r') declares the func- tion g(r) == sin(r). The shorthand command g=inline(expr) implicitely as- sume that expr is a function of the default variable x. Once an inline function has been declared, it can be evaluated at any set of variables through the command feval. For instance, to evaluate g at the points z= [0 1] we can feval write

> > feval(' g' ,z);

We note that, cOl1trarily to the case of the cOlnrrland eval, with feval the name of the variable (z) needs rlot coirlcide with the syrrlbolic name (r) assigned by the irlline conlmand.

After this qllick irltrodllctioll, our suggestion is to explore l'vlATLAB using the comlnal1d help, a/nd get acquainted with tlle implerrlentation of various algorithrns by the progranls described throughout this book.

:

.....:f:::i . «'." . .. '..:.....-:.:.. ,.tit

See the Exercises 1.8-1.11.

A systematic discussion on floating-point nurnbers cal1 be found in [Ueb97] or in [QSSOO]. For matters concerning the issue of complexity, we refer, e.g., to [Pan92]. For a more systematic introductioll to MATLAB the interested reader can refer to the MATLAB manual [HHOO] as well as to specific books such as [HLROl] or [EKH02].

Exercise 1.1 How many numbers belorlg to the set IF(2, 2, -2, 2)? What is the value of El'vf for such set?

Exercise 1.2 Show that the set IF(tJ, t, L, U) contains precisely 2(tJ-l)tJt-1 (U- L + 1) elen1ents.

Exercise 1.3 Write the MATLAB instructions to build an upper (reSI)eC- tively, lower) triangular matrix of dimel1sion 10 llaving 2 011 tlle mairl diagonal and - 3 on the II pper (respectively

lower) diago11al.

Exercise 1.4 Write the MATLAB instructions which allow the interchange of the third and seventh row of the matrices built up in Exercise 1.3, and then the instructions allowing the interchange between the fourth and eighth column.

Exercise 1.5 Verify whether the following vectors in

4 are linearly indepen- dent:

VI == [0 1 0 1], V2 == [1 2 3 4], V3 == [1 0 1 0], V 4 == [0 0 1 1].

Exercise 1.6 Write tlle following furlctions and corIlpute their first and sec- ond derivatives, as well as their prirnitives, using the syml)olic tc)olbox of lVIAT- LAB:

g(x) == v x2 + 1, f(x) == sin(x: 3) + cosh(x).

Exercise 1. 7 For arlY giverl vector v of ciirnensiorl n

llsirlg the comrnarld c=poly(v) orle carl cOllstruct tlle r

+ 1 coefficients of tlle polyrlomial p(x) == poly

+

c(k)xn+l-k whicil is eqllal to l1

=l(X - v(k)). In exact aritllrnetics, one sllolllld firld that v = roots(poly(c)). However, this cannot occur dlle to rOlll1doff errors, as one can check by using tlle comIlland roots (poly ([1 : nJ)), where n rarlges froll1 2 to 25.

Exercise 1.8 Write a program to cOlnpute the following sequence:

1 10 == - (e - 1), e In + 1 == 1 - (n + 1) In, for n == 0, 1, . . . ,

Compare the nUlllerical result with the exact lirnit In ---t 0 for n ---t 00.

Exercise 1.9 Explain the behavior of the sequence (1.4) wllen computed in MATLAB.

Exercise 1.10 Consider the following algorithm to COlllpllte 'if. Generate n couples {(Xk, Yk)} of random numbers in tlle irlterval [0,1], therl compute the number rr

of thern lying inside tlle first quarter of the unit circle. Obviously, 7r turns out to be the limit of the sequence 7r n == 4m/r1. Write a MATLAB program to COll1pute this sequence and check the error for irlcreasing values of n.

Exercise 1.11 Write a prograln fc)r the cornplltatiorl of the binolllial coef- ficient ('Z) == 77J! / (k! (r

- k)!), wllere TZ, arld k are two natllral 11ll1111)ers with k < rL

2. Nonlinear equations

Computing the zeros of a function f (equivalently, the roots of the equa- tion f (x) == 0) is a problem that we encounter quite often in scientific computing. III general, this task canrlot be accomplished in a finite num- ber of operations. For instance, we have already seen in Section 1.4.1 that when f is a generic polynomial of degree greater than 4, there do not exist closed formulae for the zeros. The situation is even more difficult when f is not a polynomial. Iterative methods are therefore adopted. Starting from one or several initial data, the methods build up a sequence of values x(k) that hopefully will converge to a zero a of the function f at hand.

Problem 2.1 (Investment fund) At the beginning of every year a bank customer deposits v euros in an investment fUIld and withdraws, at the end of the n-th year, a capital of M euros. We want to compute

the average yearly rate of interest 1 of this investment. Since 1\1 is related to I by the relation

k 1+1 n M=vL.,.,(I+1) =v I [(1+1) -1], k==I we deduce that 1 is the root of the algebraic equation: f(1) = 0 where f(1) = A1 - v 1 ; I [(1 + It - 1]. This problen1 will be solved in Example 2.1.

.

Problem 2.2 (State equation of a gas) We want to determine the volume V occupied by a gas at temperature T and pressure p. The state equation (i.e. the equation that relates p, V and T) is

[p + a(NjV)2] (V - Nb) == kNT,

(2.1)

where a arid b are two coefficients that depend on the specific gas, N is the number of molecules which are contained in the volume V and k is the Boltz- mann constant. We need therefore to solve a nonlinear equation whose root is V (see Exercise 2.2). .

Problem 2.3 (Statics) Let us consider the nlechanical system represented by the four rigid rods ai, of Figure 2.1. For any admissible value of the angle {3, let us determine the value of the corresponding angle 0: between the rods al and a2. StartiIlg frorn tIle vector identity

al - a2 - a

- a4 == 0

and lloting tIlat the rod al is always aligned with the x-axis, we can deduce the following relationship between (3 and ex: al al . a

+ a

- a

+ a

- cos({3) - - cos(ex) - cos({3 - 0:) == - , (2.2) a2 a4 2a2a4 vvhere Q'i is the known length of the i-th rod. This is called the Freudenstein equation, and we can rewrite it as follows: f(o:) == 0, where f(x) == (al/a2) cos({3) - (al/a4) cos(x) - cos({3 - x) + (a

+ a

- a

+ a

)/(2a2a4). A solution in explicit form is available only for special values of {3. We would also like to mention that a, solution does not exist for all values of {3, and may not even be unique. To solve the

equation for any given {3 lying between 0 and 1[' we should invoke numerical methods (see Exercise 2.9). .

y

x

Fig. 2.1. System of four rods of Problem 2.3

Let f be a continuous function in [a, b] which satisfies f(a)f(b) < o. Then necessarily f has at least one zero in (a, b). Assume for simplicity that it is unique, and let us call it Ct. (In the case of several zeros, by the help of the command fplot we can locate an interval which contains only one of them.) The strategy of the bisection method is to halve the given interval and select that subinterval where f features a sign change (this subinterval

will henceforth COlltain a). This procedure is iterated until the last selected interval is so sInal1 that we have reached the desired accuracy. The sequence {x(k)} of the midpoints of these subintervals](k) will inevitably tend to a since the length of the subintervals tends to zero as k tends to infinity.

y](0)](1) .. I I I I I I I I a(O) x(O): I

I , :](3): :-

:](2) : I ,...

.- ' ,

Fig. 2.2. A few iterations of the bisection metll0d

Precisely, the method is started by setting

a (0) == a, b(O) == b, [(0) == (a (0) , b(O)), x(O) == (a (0) + b(O)) /2.

At the step k > 1 we select the subinterval [(k) == (OJ(k), b(k)) of the interval [(k-1) == (a(k;-l), b(k-1)) as follows: given x(k-1) == (a(k-1) + b(k-1))/2, if f(x(k-1)) == 0 then a == x(k-1) and the method terminates; otherwise,

if f(a(k-1))f(x(k-1)) < 0 we set a(k) == a(k-1), b(k) == x(k-1); if f(x(k-1))f(b(k-1)) < 0 we set CL(k;) == x(k-1), b(k) == b(k;-l). Then we define x(k) == (a(k) + b(k))/2 and k is increased by 1. For instance, in the case represented in Figure 2.2, which corresponds to the choice f(x) == x 2 - 1, by taking a(O) == -0.25 and b(O) == 1.25, we would obtain

[(0) == (-0.25,1.25), [(1) == (0.5,1.25), [(2) == (0.875, 1.25), [(3) == (0.875,1.0625),

x(O) == 0.5, x(I) == 0.875, x(2) == 1.0625, x(3) == 0.96875.

Notice that every subinterval f(k) contains the zero a. Moreover, the sequence {x Ck)} necessarily converges to a since at each step the length IICk) I = b Ck) - a Ck) of ICk) halves. Since IICk) I = (Ij2)kII C O) I, the error at the k - th step satisfies

1 (I) k+I leCk) I = Ix Ck) - aI < 2 IICk) I = 2 (b - a).

In order to guarantee that Ie(k) I < E, for a given tolerance E it suffices to carry out k min iterations, being k min the least integer that satisfies the inequality

(b - a) k min > log2 E - 1.

(2.3)

Obviously, this inequality makes sense in general, and is not confined to the specific choice of f that we have made before.

The bisection method is implemented in Program 1: fun is a string (or an inline function) that specifies the function f, a and b are the extremes of the search interval, tol is the tolerance E and nmax the maximum number of iterations allotted. When fun is an inline function (or a function defined in am-file) besides the first argument relative to the independent function it can accept other auxiliary parameters needed for the definition of fun. Output parameters are zero, which contains the approximate value of a, the residual res which is the value of f in zero and ni ter which is the total number of iterations that are carried out. The command find find(fx==O) finds those indices of the vector fx corresponding to null sign components, while the command sign (fx) determines the sign of fx.

fu nction [zero, res, n iter]==bisection (fu n ,a, b, tol, nmax, varargin) %BISECTION Find function zeros. % ZERO==BISECTION(FUN,A,B,TOL,NMAX) tries to find a zero ZERO of the % continuous function FUN in the interval [A,B] using the bisection method. % FUN accepts real scalar input x and returns a real scalar value. If the search % fails an errore message is displayed. FUN can also be an inline object. % ZERO==BISECTION(FUN,A,B,TOL,NMAX,PI,P2,...) passes parameters PI, % P2,... to the function FUN(X,PI,P2,...). % [ZERO,RES,NITER]== BISECTION(FUN,...) returns the value of the % residual in ZERO and the iteration number at which ZERO was computed. x == [a, (a+b)*O.5, b]; fx == feval(fun,x,varargin{:});

if fx(1)*fx(3»0 error(' The sign of the function at the extrema of the interval must be different'); elseif fx(1) ==== 0 Z e ro == a. res == O . niter == O. , , , return elseif fx(3) ==== 0 zero == b; res == 0; niter == 0; retu rn end niter == 0; I == (b - a)*0.5; while I >== tol & niter <== nmax niter == niter + 1; if si g n (fx (1)) * s i g n (fx (2)) < 0 x(3) == x(2); x(2) == x(1)+(x(3)-x(1))*0.5; fx == feval(fun,x,varargin{:}); I == (x(3)-x(1))*0.5; elseif sign(fx(2))*sign(fx(3)) < 0 x(l) == x(2); x(2) == x(1)+(x(3)-x(1))*0.5; fx == feva I(fun ,x, varargin {:}); I == (x(3)-x(1))*0.5; else x(2) == x(find(fx====O)); 1==0; end end if niter> nmax fprintf(['bisection stopped without converging to the desired tolerance',... 'because the maximum number of iterations was reached\n']); end zero == x(2); x == x(2); res == feval(fun,x);

Example 2.1 Let us apply the bisection method to solve Problem 2.1, as- suming that v is equal to 1000 euros and that after 5 years M is equal to 6000 euros. The graph of the function f can be obtained by the following instructions

» f==inline('M-v*(1+1).*((1+1).A5 - l)./l','l','M','v'); > > fplot(f, [0.01,0.3], [], [], [] ,6000, 1000)

We see that f has a unique zero in the interval (0.01,0.1), which is about equal to 0.06. If we execute the Program 1 with tol== 10- 12 , a== 0.01 and b== 0.1, after 36 iterations the method converges to the value 0.061402, in perfect agreement with the estimate (2.3) according to which k min == 36. We thus conclude that the interest rate I is approximately equal to 6.14%.

In spite of its simplicity, the bisection method does not guarantee a monotone reduction of the error, but simply that the search interval is

halved from one iteration to the next. Consequently, if the only stopping

criterion that is adopted is the control of the length of [(k), one might discard approximations of a which are quite accurate. As a matter of fact, this rnethod does not take into proper account the actual behavior of f. A striking fact is that it does not converge in a single iteration even if f is a linear function (unless the zero a is the Inidpoint of the initial search interval).

:

..:..:.::.:'j . .'....., ..., ::::=:::::::-:.; ..... /" . :::

:?-:.:.:.:.:.?:::::::::::}::::::':

See Exercises 2.1-2.5.

A more efficient method for the computation of the zeros of a function f can be constructed by exploiting the differentiability of f. In that case,

$$y(x) = f(x(k)) + f'(x(k))(x - x(k))$$

provides the equation of the tangent to the curve (x, f (x)) at the point x(k) . If we pretend that x(k+l) be such that y(x(k+l)) == 0, we obtain:

X(k+l) = x(k) - :,i:

))) , provided f' (x(k)) t= 0

(2.4)

This formula allows us to compute a sequence of values x(k) starting from an initial guess x(O). This method is known as Newton's method and corresponds to computing the zero of f by locally replacing f by its tangent line (see Figure 2.3). As a matter of fact, by developing f in Taylor series in a neighborhood of a generic point x(k) we find that

f(X(k+l)) = f(x(k)) + <5(k) f'(x(k)) + O((<5(k))2),

(2.5)

where <5(k) == x(k+l) - x(k). Forcing f(x(k+l)) to be zero and neglecting the term O((<5(k))2), we can obtain x(k+l) as a function of x(k) as stated in (2.4). In this respect (2.4) can be regarded as an approximation of (2.5). Obviously, (2.4) converges in a single step when f is linear, that is f(x) == alX + ao; in such a case

x(1) = x(D) _ alx(D) + aD al

aO

al

2.2 The Newton method 43

8

-4

6 """ ""., ",," ",."", ,.,

4 """""""""

, ."""""" .,.. i I I ,: """""" t ' , I I i

2 '

o '

x(O)

-2 '

. . . . '" , ,.,.., J' ",..,... '_'"

-6 -2 -1.5 -1 -0.5 0 0.5

1.5 2

Fig. 2.3. The first iterations generated by the Newton method with initial guess x(O) for the function f(x) == x + eX + 10/(1 + x 2) - 5

Example 2.2 Let us solve Problem 2.1 by Newton's method, taking as initial data x(O) == 0.3. After 6 iterations the difference between two subsequent iterates is less than or equal to 10- 12 .

The Newton method in general does not converge for all possible choices of x(O), but only for those values of x(O) which are

sufficiently close to a. At a first glance, this request looks meaningless: indeed, in order to compute a (which is unknown), one should start from a value sufficiently close to a! In practice, a possible initial value x(O) can be obtained by resorting to a few iterations of the bisection method or, alternatively, through an investigation of the graph of f. If x(O) is properly chosen and a is a simple zero (see Section 1.4.1) then the Newton method converges. Furthermore, in the special case in which f is continuously differentiable up to its second derivative one has the following convergence result (see Exercise 2.8),

X(k+l) _ a lim k-+oo (x(k) - a)2

f" (a) 2f'(a)

(2.6)

Consequently, Newton's method is said to converge quadratically, or with order 2, since for sufficiently large values of k the error at the (k + l)-th step behaves like the square of the error at the k-th step multiplied by a constant which is independent of k. In the case of zeros with multiplicity m larger than 1, the order of convergence of Newton's method downgrades to 1 (see Exercise 2.15). In that case one could recover the order 2 by modifying the original method

Ji,.\ .\-

as follows:

X(k+1) = X(k) _ m f(x(k)) provided J / (x(k)) # O. (2.7) J/(x(k)) ,

Ob\Tiously, this modified Newton method requires the a-priori knowledge of m. If this is not the case, one could develop an adaptive Newton method, still of order 2, as described in [QSSOO], Section 6.6.2.

Example 2.3 The function f(x) == (x - 1) log(x) has a single zero a == 1 of multiplicity m == 2. Let us compute it by both Newton's method (2.4) and by its modified version (2.7). In Figure 2.4 we report the error obtained using the two methods versus the iteratiol1 number. Note that for the classical Newton's method the convergence is only linear.

10 2

10- 2

... ... """ .":, - .....

10- 4

... . , """""

,. . ", " . ..- ."" "" , ,. ". "", ." , ... , ,.. "" "" .,. . ,., . . ", """

"

",' ..,.., ,., ,,' """ .

',

10- 6 ",,,,,, ,

10- 8 """",., ,.,.

10- 10 o

5

10

15

20

25

30

Fig. 2.4. Error versus iteration number for the function of Example 2.3. Dashed line corresponds to Newton's method (2.4), solid line to the modi- fied Newton's method (2.7) (with m == 2)

In theory, a convergent Newton's method returns the zero a only after an infinite number of iterations. In practice, one requires an approxima- tion of a up to a prescribed tolerance E. Thus the iterations can be terminated at the least km,in for which the following inequality holds:

le(k rnin) I = la - x(krnin) I < E.

Unfortunately, since the error is unknown, one needs to employ in its place a suitable error estimator, that is, a quantity that can be easily computed and through which we can estimate the real error. At the end of Section 2.3, we will see that a suitable error estimator for Newton's method is provided by the difference between two successive iterates. This means that one terminates the iterations at the kmin-th step as soon as

IX(k rnin) _ x(krnin -1) I < E.

(2.8)

Remark 2.1 (The general case) The stopping criterion (2.8) may not be suitable for methods different from Newton's. Alternatively, one could use an error estimator based on the residual at the step k defined as r Ck) == f(x Ck)) (note that the residual is null when x Ck) is a zero of the function f). We could stop the iteration at the first k rnin for which

If(xCkm,in))1 < E.

Using the residual as an error estimator is satisfactory only when the behavior of f is almost linear in a neighborhood la of the zero a (see Figure 2.5). Otherwise, it will produce an over estimation of the error if I f' (x) I » 1 for x E I a and an under estimation if I f' (x) I « 1 (see also Exercise 2.6).

y

f

y

f(X(k))

f

a

e(k)

Fig. 2.5. Two situations in which the residual is a poor error estimator: If'(X)1 » 1 (left), If'(X)1 « 1 (right), with x belonging to a neighborhood of a

In Program 2 we implement Newton's method (2.4). Its modified form can be obtained simply by replacing f' with f'lm. The input parameters f and df are the strings which define the function f and its first deriva- tive, while xO is the initial guess. The method will be terminated when the absolute value of the difference between two subsequent iterates is less than the prescribed tolerance tal, or when the maximum number of iterations nmax has been reached.

fu nction [zero, res, n iter] == newton (f, df,xQ, tol, n m ax, va ra rgi n) %NEWTON Find function zeros. % ZERO==NEWTON(FUN,DFUN,XQ,TOL,NMAX) tries to find the zero ZERO of % the continuous and differentiable function FUN nearest to XQ using the Newton

% method. FUN and its derivative DFUN accept real scalar input x and returns % a real scalar value. If the search fails an errore message is displayed. % FUN and DFUN can also be inline objects. % ZERO==NEWTON(FUN,DFUN,XO,TOL,NMAX,P1,P2,...) passes parameters % P1,P2,... to functions: FUN(X,P1,P2,...) and DFUN(X,P1,P2,...). % [ZERO,RES,NITER]== NEWTON(FUN,...) returns the value of the % residual in ZERO and the iteration number at which ZERO was computed. x == xO; fx == feva l (f, x, va r a rg in { : }); d fx == feva l (d f, x, va r a rg in { : }); niter == 0; d iff == to 1+ 1 ; while diff >== tol & niter <== nmax niter == niter + 1; diff == - fx/ dfx; x == x + d iff; d iff == a b s (d iff) ; fx == feva l (f, x, va r a rgi n { : }); d fx == feva l (d f, x, va r a rg in { : }); end if niter> nmax fpri ntf([' newton stopped without convergi ng to the desi red tolera nce' ,... 'because the maximum number of iterations was reached\n']); end zero == X' res == fx' , ,

. -; . -.. . . ..

1. Methods for the computation of the zeros of a function f are usually of iterative type;

2. the bisection method complltes a zero of a function f by generating a sequence of intervals whose length is halved at each iteration. This method is convergent provided that f is continllous ill the initial interval and has opposite signs at the end-points of this interval;

3. Newton's method computes a zero a of f by taking into account the values of f and of its derivative. A necessary condition for convergence is that the initial datum belongs to a suitable (sufficiently small) neighborhood of a;

4. Newton's method is quadratically convergent only when a is a simple zero of f, otherwise convergence is linear.

2.3 Fixed point iterations

See Exercises 2.6- 2.14.

.. . .

J,"'" :- . ...ii . \. .. .:.;. . . . . . " .... " ....., .'.

Playing with a pocket calculator, one may verify that by applying repeat- edly the cosine key to the real vallle 1, one gets the following

sequence of real numbers:

X(l) == cos(l) == 0.54030230586814, x(2) == COS(X(l)) == 0.85755321584639, X(lO) == cos(x(9)) == 0.74423735490056, x(20) == coS(x(19)) == 0.73918439977149,

which should tend to the value a == 0.73908513. ... Since, by construc- tion, x(k+l) == cos(x(k)) for k == 0,1,... (with x(O) == 1), the limit a satisfies the equation cos(a) == a. For this reason a is called a fixed point of the cosine function. We may wonder how such iterations could be exploited in order to compute the zeros of a given function. In the previous example, a is not only a fixed point for the cosine function, but also a zero of the function f (x) == x - cos(x), hence the rnethod previ- ously proposed can be regarded as a method to compute the zeros of f. On the other hand, not every function has fixed points. For instance, by repeating the previous experiment using now the exponential function and still x(O) == lone encounters a situation of overflow after 4 steps only (see Figure 2.6).

y

1 .

1 .

y

2

1.5 ....

0.5 ..

0.5 .",.."".""

O'

x

O'

x

a

-0.5 ..,.

-0.5 -0.5

o

0.5

1.5

-1 ' , 2 -1 -0.5 a 0.5

1.5 2

Fig. 2.6. The function cp(x) == cos x admits one and only one fixed point (left), whereas the function cp(x) == eX does not have any (right)

Let us clarify the above intuitive idea by considering the following problem. Given a function cjJ : [a, b] -+ JR, find Q E [a, b] such that Q == cjJ(Q).

If such an Q exists it will be called a fixed point of cjJ and it could be computed by the following algorithm:

X(k+l) == cjJ(x(k)), k > 0

(2.9)

where x(O) is all initial guess. This algorithm is called fixed point itera- tions and cjJ is said to be the iteration function. The introductory example is therefore an instance of fixed point iterations with cjJ(x) == cos(x). A geometrical interpretation of (2.9) is provided in Figure 2. 7 (left). One can guess that if cjJ is a continuous function alld the limit of the sequence {x(k)} exists, then such limit is a fixed point of cjJ. We will make this result more precise in Proposition 2.1 and 2.2.

Example 2.4 The Newton method (2.4) can be regarded as an algorithm of fixed point iterations whose iteration function is

f(x) cp(x) = x - f'(x)"

(2.10)

From now on this fUllction will be denoted by CPN (where N stands for Newton). This is not the case for the bisection method since the generic iterate x(k+l) depends not only on X(k) but also on x(k-l).

As shown in Figure 2.7 (right), fixed point iterations may not converge. Indeed, the following result holds.

Proposition 2.1 Assume that the iteration function in (2.9) satisfies the following properties: 1. cjJ(x) E [a,b] for all x E [a,b]; 2. cjJ is differentiable in [a, b]; 3. 3K < 1 such that IcjJ'(x)/ < K for all x E [a, b]. Then cp has a unique fixed point Q E [a, b] and the sequence defined in (2.9) converges to Q, whatever choice is made for the initial datum x(O) in [a, b]. Moreover

X(k+l) _ Q lim (k) = cjJ' (a) k-HX) X - Q

(2.11)

2

2

1 ,..

(2) ,x.. I .. ..

..:I..x x{O)

y

1 ' , ,

1.5 "

1.5' "

0.5 "

0.5 "

0,

x a " x(O) x(2) a x(I) -0.5 a 0.5 1.5 2 -0.5 a 0.5 1.5 2

-0.5 -0.5

Fig. 2.7. Representation of a few fixed point iterations for two different itera- tion functions. To the left, the iterations converge to the fixed point a, whereas the iterations on the right produce a divergence sequence

From (2.11) one deduces that the fixed point iterations converge at least linearly, that is, for k sllfficiently large the error at the step k + 1 be- haves like that at the k step multiplied by a constant cjJ' (a) which is independent of k and whose absolute value is strictly less tllan 1.

Example 2.5 The function cjJ(x) == cos(x) satisfies all the assumptions of Propositioll 2.1. Indeed, IcjJ'(a)I == I sin(a)1 r'-I 0.67 < 1, and thus by continuity there exists a neighborhood lex of a such that IcjJ'(x) I < 1 for all x E lex. The function cjJ(x) == x 2 - 1, although possessing two fixed points a=:t == (1:i: J5) /2, does not satisfy the assumption for either since IcjJ'(a=:t)I == 1 1 :i: V51 > 1. The corresponding fixed point iterations will not converge.

The Newtoll method is not the OIlly iterative procedure featuring quadratic convergence. Indeed, the following general property holds.

Proposition 2.2 Assume that all hypotheses of Proposition 2.1 are sat- isfied. In addition ass'ume that cjJ is differentiable twice and that

cjJ' (a) == 0, cjJ" (a) =f O.

Then the fixed point iterations (2.9) converge with order 2 and

X(k+l) - a 1 lim (k))2 = -(j/' (ex). k-+oo X - a 2

(2.12)

Example 2.4 shows that the fixed point iterations (2.9) could also be used to compute the zeros of the function f. Clearly for any given f the function cjJ defined in (2.10) is not the only possible iteration function.

For illsta!nce

fc)r tile SC)llltic)ll c)f tIle eC]llatic)ll lc)g(x) ==

/

after settiIlg .f(x) == lc)g(:1') - r

tIle ClIC)ice (2.1()) cC)llllcIleacl tC) tIle iteratic)ll flillctic)ll

cP IV (

r) == :1' (1 - 1 C) g (:1') +

}).

Allotller fixecl poirlt iteratic)ll aJgc)ritllllll cC)lllcll)e ()1)taillecl by c},clclillg :1' tC) l)otil sicles ()f tIle eClllatic)ll f (.f) == o. TIle assc)ciated i teratiC)ll f\lllCtic)ll is 110W cPl (

r) == .1' + lc)g (

r) - r. A fl1rtller llletJlC)cl CC)lllcl 1)e C) l)tclilleci l)

y ClIC)osillg tIle i teratiC)ll fllllCtic)ll 02 (Jo) == Jol()g(r) /

'y. N C)t all tilese llletll()cls aTe CC)llvergellt. For illstallce

if

/ == - 2, tile llletll0ds cc)rreSpOllclillg tC) tIle iterati()ll fllllCtic)llS (pJ.ry aJld CP2 are lJotll C()llvergerlt

wilereas tIle C)lle witil iteratioll fUIICtic)ll cPl is ll()t sillce l cP; (x) l > 1 ill a, 11eighl)orlloc)cl of tIle fixed !-)Oillt Ct.

In general, fixecl POilIt iteratiolls aTe terlrlillated whel1 tIle absoll1te va111e of the differellce betweell two cOllseclltive iterates is less thall a'l)rescril)ed tolerallce E. Sillce a == cjJ(a) and

r(k+l) == cjJ(x(k))

llsirlg tIle lileall va.llle tileorenl (see Sectioll 1.4.3) we find

0: - x(k+l) = cjJ(a) - cjJ(.y(k)) = cjJ'(

(k)) (0: - :r(k)) with

(k) E Ia,:r(l.),

10'.:1:(1,:) being tIle illter"val witil eIld-I)oirlts Ct arId :I

(k). USiIlg tIle iclelltity

Q _ x(k) == (Q _

1'(k+ 1)) + (:r(k+l) _ .r(k))

it follc)ws tilat

a - :r (k) = 1 _ q); (

(k)) (,r (k + 1) - :r (k)) .

(2.13)

CC)llSeCIlleIltly

if QI (.1')

() ill a lleiglll)c)rllC)()d of (), tlle clifferellce l)etweell t\\TO CC)llSecl1ti\le iterates lJr(Jvicles a satisfactc)ry" error estilllat()r. Tilis is tlle case fc)r 111etllC)cls ()f ()rcler 2

illCllllClillg N ewtC)ll

s llletllC)cl. TIlis estilnate becolIles as l11C)re UIlsatisfactor

y as c/JI approaclles 1.

Example 2.6 Let lIS COlllpute witil Newtol1'S 111etilod tlle zero a == 1 of tlle flll1Ctioll f (:r) == (x - 1) nz -1 log (x) fc)r rrL == 11 alld 711 == 21

wilose 111ultiplicity is equal to 71Z. III tilis case N ewtoll

s llletlloci COllverges with order 1; llloreover

it is p()ssible to prove (see Exercise 2.15) tllat cp

(a) == 1-l/m

CPN being the iteration function of tlle method, regarded as a fixed point iteration algorithrrl. As m increases, the accuracy of the error estilllate furllis11ed by the difference

betweell two conseclltivc iterates decreases.1-'11is is cOllfirnlCcll)y tlle llllllllcrical resllits ill

'igllre 2.8 wllere we COl111)are tlle l)(=

llavior of tIle trlle error witll tllat of Ollr estilllator for l)()tll ,(IL == 11 alle1 Tn! == 21. TIle (lifferellce l)etweell tllese two clllalltities illereases as rn, illcreases.

10°

10? ,

" , " , .. , , , , , , , , , , " , , (1) "" , , , , , , ,

10 4

1 0. 6

" "

" "

" .....

10- 8

" "

" "

" "

10 10

10 12 o 50 100 150 200 250 300 350 400

Fig. 2.8. Absc)lute val lles ()f the errors (solid line) and al)sollite values of tile differellce [)etweell two cOllseclltive iterates (dashed line), plotted verSllS tlle 11unlber of iterations f()r tlle case of EXalllll)le 2.6. Graphs (1) refer to TIL == 11, graphs (2) to rrL == 21

1. A ll11111ber a satisfyillg 1Y((}) == () is caJ1ed a fixed P()illt of cPo F()r its C0111p11tati()11 we call 11se tlle so-ca11e(1 fixed l) ()i11t iterati011S: x(k+l) == 1Y(:r(k))

2. fixec} l)()illt i terati()llS COllvergc 1111(1e1' S11i t(1)1e assl1llll)ti()11S ()ll (P al1e1 its first e1e1'ivative. T:VJ)ical1)l, C{)ll\rergellce is 1illf'ar

ll()\\VeVer

ill tlle SI)ecial case wllell 0

' (cr) == (), tlle fixec1 I)()i11t i teratiC)I1S C()llverge q 11 a!c1r at i c (11)l :

3. fixecl I)c)illt it erati C) llS call })(' 11Se(1 alsc) t() C()111I)11te tlle zer()s ()f a f1111Cti()11.

See Exercises 2.15-2.18.

The nl0st SOpllisticated 111etllods for tlle COlllputatioll of tlle zeros of a fUllction c01nbille differellt algorithnls. Ill partiCtllar, tlle

lATL

;\B f1111C-

. .:::::::.

:

ftJ;:::. ... . : . :: . :.:..... -: . . . : . ...:.:.:.:.:.. .:.:.:.. .

,.,,

:-:::#'. " . ..:... ....)

..

'..LJ:.

fzero tion fzero (see Section 1.4.1) adopts the so called Dekker-Brent method (see [QSSOO], Section 6.2.3). In its basic form fzero(fun,xO) computes the zero of the function fun, where fun can be either a string which is function of x, or the name of an inline function or the name of am-file. For instance, we could solve the problem of Example 2.1 also by fzero, using the initial value xO=O. 3 (as done by Newton's method) through the following instructions: » f==inline('6000 - 1000*(1+I)jl*((1+I)A5 - 1)','1'); xO==O.3; > > [a I pha, res, flag, iter] ==fzero(f,xO); we obtain alpha=O. 06140241153653 with residual res=9. 094ge-13 in i ter=29 iterations. When flag is negative it means that fzero cannot find the zero. The Newton method converges in 6 iterations to the value 0.06140241153652 with a residual equal to 2.3646e-11. In this chapter, we have frequently mentioned the problem of how to compute the zeros of a polynomial. The solution of this problem requires ad hoc algorithms which are typically based on a deflation technique, a procedure which eliminates automatically the zeros already computed. Among others let us mention the Sturm method, the Muller method, the Newton-Horner method (see [Atk89] or [QSSOO]) and the Bairstow method ([RR85]). A different approach consists of characterizing the ze- ros of a function as the eigenvalues of a special matrix (called the com- panion matrix) and then using appropriate techniques for their compu- tation. This approach is adopted by the MATLAB function roots which has been introduced in Section 1.4.2. Newton's method and fixed point

iterations can be easily extended to compute the roots of nonlinear systems of the following form (see [QSSOO], Chapter 7):

11 (Xl, X2,..., X n) == 0, 12(Xl,X2,...,X n) == 0,

I n (Xl, X2, · · · , X n) == 0, where 11,. · · , In are nonlinear functions. Other methods exist as well, such as the Broyden and quasi-Newton methods, and can be regarded as generalizations of Newton's method (see [DS83]). The MATLAB instruction zero == fsolve('fun', xO) allows the computation of one zero of a nonlinear system defined through the user function fun starting from the vector xO as initial guess. The

function fun returns the n values !l(X),..., !n(x) for any value of the input vector x. For instance, let us consider the following system:

{ x2 + y2 == 1, sin(7rx/2) + y3 == 0, whose solutions are (0.4761, -0.8794) and (-0.4761,0.8794). The corresponding MATLAB user function, which we call systemnl, is defined as follows:

(2.14)

function fx==systemnl(x) fx(l) == x(1)A2+x(2)A2-1; fx(2) == sin(pi*0.5*x(1))+X(2)A3;

The MATLAB instructions to solve this system are therefore: » xO == [1 1]; > > a l pha ==fsolve(' system n l' ,xO) alpha == 0.4761 -0.8794

Using this procedure we have found only one of the two roots. The other can be computed starting from the initial datum -xO.

Exercise 2.1 Given the function f (x) == cosh x + cos x -

, for

== 1, 2, 3 find an interval that contains the zero of f. Then compute the zero by the bisection method with a tolerance of 10- 10 .

Exercise 2.2 For carbon dioxide (C0 2) the coefficients a and b in (2.1) take the following values: a == 0.401Pa m 6 , b == 42.7 · 10-6m 3 (Pa stands for Pascal). Find the volume occupied by 1000 molecules of CO 2 at a temperature T == 300K and a pressure p == 3.5 . 10 7 Pa by the bisection method, with a tolerance of 10- 12 (the Boltzmann constant is k == 1.3806503.10- 23 JouleK- 1).

Exercise 2.3 An object is standing on a plane whose slope varies with con- stant velocity w. After t seconds its position is

s(t,w) = 2

2 [sinh(wt) - sin(wt)],

where 9 == 9.8mj 8 2 denotes the gravity acceleration. Assuming that this object has moved by 1 meter in 1 second, compute the corresponding value of w with a tolerance of 10- 5 .

Exercise 2.4 Prove inequality (2.3).

Exercise 2.5 Motivate why in Program 1 the instruction x(2) = x(1)+ (x(3)- x(l)) *0.5 has been used instead of the more natural one x(2)= (x(1)+x(3)) *0.5 in order to compute the midpoint.

Exercise 2.6 Apply Newton's method to solve Exercise 2.1. Why is this method not accurate when

== 2?

Exercise 2.7 Apply Newton's method to compute the square root of a pos- itive number a. Proceed in a similar manner to compute the cube root of a.

Exercise 2.8 Assuming that Newton's method converges, show that (2.6) is true when a is a simple root of f(x) == 0 and f is twice

continuously differentiable in a neighborhood of a.

Exercise 2.9 Apply Newton's method to solve Problem 2.3 for {3 E [0, 21T /3] with a tolerance of 10- 5 . Assume that the lengths of rods are a1 == 10 cm, a2 == 13 cm, a3 == 8 cm and a4 == 10 cm. For each value of (3 consider two possible initial data, xeO) == -0.1 and xeO) == 21T /3.

Exercise 2.10 Notice that the function f(x) == eX - 2X2 has 3 zeros, a1 < 0, a2 and a3 positive. For which value of xeO) does Newton's method converge to a1 ?

Exercise 2.11 Use Newton's method to compute the zero of f(x) == x 3 - 3x 2 2- x + 3x4- x - 8- x in [0,1] and explain why convergence is not quadratic.

Exercise 2.12 A projectile is ejected with velocity vo and angle a in a tunnel of height h and reaches its maximum range when a is such that sin(a) == V 2gh/v5, where 9 == 9.8 m/s2 is the gravity acceleration. Compute a using Newton's method, assuming that vo == 10 m/s and h == 1 m.

Exercise 2.13 Solve Problem 2.1 by Newton's method with a tolerance of 10- 12 , assuming M == 6000 euros, v == 1000 euros and n == 5. As an initial guess take the result obtained after 5 iterations of the bisection method applied on the interval (0.01,0.1).

Exercise 2.14 A corridor has the form indicated in Figure 2.9. The maximum length L of a rod that can pass from one extreme to the other sliding on the ground is given by

L == I2/(sin(1T -

- a)) + Il/ sin(a),

where Q is the solution of the nonlinear equation

I cos(7r -')' - a) -I cos(a) = O. 2 sin 2 (7r _ ')' _ a) 1 sin 2 (a)
Compute Q by Newton's method when I2 == 10, Il == 8 and

== 37r /5.

(2.15)

Il

Fig. 2.9. The problem of a rod sliding in a corridor

Exercise 2.15 Let CPN be the iteration function of Newton's method when regarded as a fixed point iteration. Show that cp

(Q) == 1 - l/m where Q is a zero of f with multiplicity m. Deduce that Newton's method converges quadratically if Q is a simple root of f(x) == 0, and linearly otherwise.

Exercise 2.16 Deduce from the graph of f(x) == x 3 + 4x 2 - 10 that this function has a unique real zero Q. To compute Q use the following fixed point iterations: given x CO), define X Ck + 1) such that

X(k+l) = 2(x(k))3 + 4(x(k))2 + 10 3 (xC k)) 2 + 8x C k) ,

k>O

and analyze its convergence to Q.

Exercise 2.17 Analyze the convergence of the fixed point iterations

X(k+1) = x(k) [(x(k))2 + 3a] 3(X Ck))2 + a '

k > 0,

for the computation of the square root of a positive number a.

Exercise 2.18 Repeat the computations carried out in Exercise 2.11 using now the stopping criterion based on the residual (see Remark 2.1). Which is the more accurate result?

3. Approximation of functions and data

Approximating a function f consists of replacing it by another function f of simpler form that may be used as its surrogate. This strategy is used frequently in numerical integration where, instead of computing J: f(x)dx, one carries out the exact computation of J: J(x)dx, J being a function simple to integrate (e.g. a polynomial), as we will see in the next chapter. In other instances the function f may be available only partially through its values at some selected points. In these cases we aim at constructing a continuous function f that could represent the empirical law which is behind the finite set of data. We provide a couple of examples which illustrate this kind of approach.

Problem 3.1 (Climatology) The air temperature near the ground depends on the concentration K of the carbon acid therein. In Table 3.1 (taken from Philosophical Magazine 41, 237 (1896)) we report for 3 different values of K the variatioll of the average temperature with respect to the actual average temperature (normalized at the reference value K == 1) at different latitudes on the Earth. In this case we can generate a function that, on the basis of the available data, provides an approximate value of the average temperature at any possible latitude and for other values of K (see Example 3.1). .

Problem 3.2 (Finance) In Figure 3.1 we report the price of a stock at the Zurich stock exchange over two years. The curve was obtained by joining with a straight line the prices reported at every day's closure. This simple representation indeed implicitly assumes that the prices change linearly in the course of the day (we anticipate that this approximation is called composite linear interpolation). We ask wheter from this graph one could predict the stock price for a short time interval beyond the time of the last quotation. We will see in Section 3.4 that this kind of prediction could be guessed by

Latitude K == 0.67 K == 1.5 K == 2.0 K == 3.0 65 -3.1 3.52 6.05 9.3 55 -3.22 3.62 6.02 9.3 45 -3.3 3.65 5.92 9.17 35 -3.32 3.52 5.7 8.82 25 -3.17 3.47 5.3 8.1 15 -3.07 3.25 5.02 7.52 5 -3.02 3.15 4.95 7.3 -5 -3.02 3.15 4.97 7.35 -15 -3.12 3.2 5.07 7.62 -25 -3.2 3.27 5.35 8.22 -35 -3.35 3.52 5.62 8.8 -45 -3.37 3.7 5.95 9.25 -55 -3.25 3.7 6.1 9.5

Tab. 3.1. Variation of the average yearly temperature on the Earth for dif- ferent values of the concentration K of carbon acid at different latitudes

resorting to a special technique known as least squares approximation of data (see Example 3.8). .

novOO

may01

novO 1

may02

Fig. 3.1. Price variation of a stock over two years

It is known that a function f can be successfully replaced in a given interval by its Taylor polynomial, which was introduced in Section 1.4.3. This technique is computationally expensive since it requires the knowl- edge of f and its derivatives up to the order n (the polynomial degree) at a given point Xo. A further drawback is represented by the fact that the Taylor polynomial may fail to accurately represent f far enough from the point Xo. For instance, in Figure 3.2 we compare the behavior of f(x) == l/x with that of its Taylor polynomial of degree 10 built around the point Xo == 1. This picture also shows the graphical interface of the

MATLAB function taylortool which allows the computation of Taylor's taylor" polynomial of arbitrary degree for any given function f. The

agreement between the function and its Taylor polynomial is very good in a small neighborhood of Xo == 1 while it becomes unsatisfactory when x - Xo gets large. Fortunately, this is not the case of other functions such as the exponential function which is approximated quite nicely for all x E IR by its Taylor polynomial related to Xo == 0, provided that the degree n is sufficiently large.

3

1

, , , , , I I I , , , ..' -

2

o 1

1.2

1.4

1.6

1.8

2

2.2 2.4 2.6 2.8

3

2 3 4 5 6 7 10 T (x) = 2-x+(x-1) -(x-1) +(x-1) -(x-1) +(x-1) -(x-1) +...+ (x-1) 10

tx

:

t:::::j . ::::::]a :::::1:

I . :::..: 1'Lt

rc::.....']

(

:

' >:::.>::: ., :qhX!

. .....-_.-. ".. .fj"""""-';;';""""""";-"w.,:: :i..... "..

/-

..... 1 .:

I

J::::;:::

:t

:::r

:

:::::

. ..:itt¥li.:

:

:J

:

:

:

:

. -- .-..... .... 1 . ..III

I!IIIII

III!.IjIIIIIIIIIIIII I.

Fig. 3.2. Comparison between the function f(x) == l/x (solid line) and its Taylor polynomial of degree 10 related to the point Xo == 1 (dashed line). The explicit form of the Taylor polynomial is also reported

In the course of this chapter we will introduce approximation methods that are based on alternative approaches.

As seen in Problems 3.1 and 3.2, in several applications it may happen that a function is known only through its values at some given points. We are therefore facing a (general) case where n + 1 couples {Xi, f(Xi)}, i == 0, . . . , n, are given; the points Xi are all distinct and are called nodes. For instance in the case of Table 3.1, n is equal to 12, the nodes Xi are the values of the latitude reported in

the first column, while the f(Xi) are the corresponding values (of the temperature) in the remaining columns.

In such a situation it seems natural to require the approximate function f to satisfy the set of relations

f(Xi) == f(Xi), i == 0,1,. . . , n

(3.1)

Such an f is called i1'Lterpolant of f and equations (3.1) are the interpo- lation conditions. Several kinds of interpolarlts could be envisaged, such as:

- polynornial interpolant: 1 (x) = ao + al x + a2x2 + . . . + anx n ;

- trigonometric interpolant:

f - ( ) -iM x + + + + iM x X == a-Me · · · ao · . . aMe

where M is an integer equal to n/2 if n is even, (n - 1)/2 if n is odd, and i is the imaginary unit;

- rational interpolant:

k 1 (x) = ao + a 1 x + · · · + a k x . ak+l + ak+2 X + . · · ak+rL-lX n

For simplicity we only consider those interpolants which depend linearly on the n + 1 unknown coefficients ai. Both polynomial and trigono- metric interpolation fall into this category, whereas the rational inter- polant does not.

Let us focus on the polynomial interpolation. The following result holds:

Proposition 3.1 For any set of couples {Xi, !(Xi)}, i == 0,..., n, with distinct nodes Xi, there exists a unique polynomial of degree less than or equal to n, which we indicate by lInf and call interpolating polynomial of the values f(Xi) at the nodes Xi, such that

lIn!(Xi) == f(Xi), i == 0, . . . , n

(3.2)

In tlLe case where the f(Xi), i == 0, . . . , n, represent the values attained by a continuous function f, llrLf is called interpolating polynomial of f (in short, interpolant of f).

To verify uniqueness we proceed by contradiction and suppose that there exist two polynomials of degree n, llnf and ll

f, both satisfying the nodal relation (3.2). Their difference, llnf -ll

f, would be a polyno- mial of degree n which vanishes at n + 1 distinct points. Owing to a well known theorem of Algebra, such a polynomial should vanish identically, and then ll

f must coincide with llnf. In order to obtaill an expression for llnf, we start from a very special case where f(Xi) vanishes for all i apart from i == k (for a fixed k) for which f(Xk) == 1. Then setting <Pk(X) == llnf(x), we must have (see Figure 3.3)

<Pk E Pn,

tpk(Xj) = 8jk = {

if j == k, otherwise

(6 j k is the Kronecker symbol).

0.8

0.6 "....,...,..

0.4 ...,....."

,,0 ...."..

-0.6 o 0.25 0.5 0.75

1.25 1.5 1.75 2

Fig. 3.3. The polynomial 'P2 E JID 4 associated with a set of 5 equispaced nodes

The functions <Pk can be written as follows:

$$n \text{ II } x-x. \quad <Pk(X) == J \text{ , } Xk - x. \quad j=O \; J \; j:i=k$$

k == 0, . . . , n.

(3.3)

We move now to the general case where {f(Xi), i == 0, . . . , n} is a set of arbitrary values. Using an obvious superposition principle we can obtain

the following expression for IInf:

n

TIn!(X) = L!(Xk)<Pk(X) k==O

(3.4)

Indeed, this polynomial satisfies the interpolation conditions (3.2), since

n

n

TIn!(Xi) = L!(Xk)'Pk(Xi) = L!(Xk)8 ik = !(Xi), i = 0, · · . , n. k==O k==O

Due to their special role, the functions 'Pk are called Lagrange charac- teristic polynomials, and (3.4) is the Lagrange form of the interpolant. In MATLAB we can store the n+1 couples {(Xi, f(Xi))} in the vectors polyfi t x and y, and then the instruction c=polyfi t (x, y ,n) will provide the coefficients of the interpolating polynomial. Precisely, c (1) will contain the coefficient of x n , c(2) that of xn-1, ... and c(n+1) the value of IInf(O). (More on this command can be found in Section 3.4.) As already seen in Chapter 1, we can then use the instruction p=polyval (c, z) to compute the value p (j) attained by the interpolating polynomial at z (j), j =1 , . . . , m, the latter being a set of m arbitrary points. In the case when the explicit form of the function f is available, we can use the instruction y=eval (f) in order to obtain the vector y of values of f at some specific nodes (which should be stored in a vector x).

Example 3.1 To obtain the interpolating polynomial for the data of Problem 3.1 relating to the value K = 0.67 (first column of Table 3.1), using only the values of the temperature for the latitudes 65, 35, 5, -25, -55, we can use the following MATLAB instructions:

> > x = [-55 -25 5 35 65]; y = [-3.25 -3.2 -3.02 -3.32 -3.1]; > > format short e; c=polyfit(x,y,4) c= 8.281ge-08 -4.5267e-07 -3.4684e-04 3.7757e-04 -3.0132e+00

The graph of the interpolating polynomial can be obtained as follows:

> > z=linspace(x(1),x(end),100); > > p=polyval(c,z); plot(z,p,x,y, '0');

In order to get a smooth curve we have evaluated our polynomial at 101 equispaced points in the interval [-55, 65] (as a matter of fact, MATLAB plots are always constructed on piecewise linear interpolation between neighboring points). Note that the instruction x(end) picks up directly the last component of the vector x, without specifying the length of the vector. In Figure 3.4 the

filled circles correspond to those values which have been used to construct the interpolating polynomial, whereas the empty circles correspond to values that have not been used. We can appreciate the qualitative agreement between the curve and the data distribution.

-2.95

- - - - - -' . . - . - . -3.05 .. . .

. . , . . , , , , , . , . .:' , ... ':' , ...,..,.

. . , . . . , , , , . . ,

. , , . . . , , . , , , . ':' , . , , . , . . . , , ... - . . . - - . - . - - . . . - - - -- - -- - - - - - -- - -- - -- - -- - - -

-3.25

-3.35 .."""" ",0..,... " ':.,.." ."..... .,. ,....., ..," ,.", ...

.,'" ,.." ,.,:" ,." ... ",. O

-

. .. - - - - - - - - - - - - - ., - . - . - - . .

-3.45 -60 -40 -20

o

20

40

60

Fig. 3.4. The interpolating polynomial of degree 4 introduced in Example 3.1

Using the following result we can evaluate the error obtained by replacing f with its interpolating polynomial lInf:

Proposition 3.2 Let I be a bounded interval, and consider n + 1 distinct interpolation nodes {Xi, i == 0,..., n} in I. Let f be continuously differentiable up to order n + 1 in I. Then '\Ix E I 3

E I such that

f(n+l)(

) n Enf(x) = f(x) - IInf(x) = (n + l)! n (x - xd (3.5)

Obviously, Enf(Xi) == 0, i == 0, . . . , n. Result (3.5) can be better specified in the case of a uniform distribution of nodes, that is when Xi == Xi-l + h for i == 1, . . . , n, for a given h > 0 and a given Xo. As stated in Exercise 3.1, '\Ix E (xo, x n) one can verify that

n hn+l IT (x - Xi) < n! 4 i==O

(3.6)

and therefore

maxlf(n+l)(x)1 maxIEnf(x) I < xEI ( ) hn+l. xEI 4 n + 1

(3.7)

,\

Unfortunately, we cannot deduce from (3.7) that the error tends to 0 when n

(x), in spite of the fact that hn+l j[4(n+ 1)] tends to o. In fact, as shown in Example 3.2, there exist functions! for which the limit can even be infinite, that is

lim max/En!(x) I == (X). n---+oo xEI

This striking result indicates that by increasing the degree n of the interpolating polynomial we do not necessarily obtain a better reconstruction of !. For instance, should we use all data of the first column of Table 3.1, we would obtain the interpolating polynomial II 12 ! repre- sented in Figure 3.5, whose behavior in the vicinity of the left-hand of the interval is far less satisfactory than that obtained in Figure 3.4 using a much smaller number of nodes. An even worse result may arise for a special class of functions, as we report in the next Example.

Example 3.2 (Runge) If the function f(x) == 1/(1 + x 2) is interpolated at equispaced nodes in the interval I == (-5,5), the error maxxEI IEnf(x) I tends to infinity when n

00. This is due to the fact that if n

00 the order of magnitude of maxxEI If(n+l) (x) I outweighs the infinitesimal order of hn+l /[4(n+ 1)]. This conclusion can be verified by computing the maximum of f and its derivatives up to the order 21 by means of the following instructions:

» syms x; n==20; f=='1/(1+x A 2)'; df==diff(f,l); > > cdf == char(df); > > for i == 1:n+1, df == diff(df,l); cdfn == char(df); x == fzero(cdfn,O); M(i) == abs(eval(cdf)); cdf == cdfn; end The absolute values of the functions f(n), n == 1,...,21, are stored in the vector M. Notice that the command char converts the symbolic expression df into a string that can be evaluated by the function fzera. In particular, the absolute values of fen) for n == 3, 9, 15, 21 are:

> > M([3,9,15,21]) == ans == 4.6686e+OO 3.2426e+05 1.2160e+12 4.8421e+19

n while the corresponding values of the maximum of IT (x - xi)/(n + 1)! are i==O

» z == linspace(-5,5,10000); » for n==O:20; h==10/(n+1); x== [-5:h:5]; c==poly(x); r(n+ 1)==max(polyval(c,z)); r(n+ 1)==r(n+ 1) / prod ([1: n+2]); end » r([3,9,15,21]) ans ==

2.8935e+OO 5.1813e-03 8.5854e-07 2.1461e-ll

where c=poly (x) is a vector whose elements are the coefficients of the polyno- poly mial whose roots are the elements of the vector x. It follows that maxxEI IEnf(x) I attains the following values:

> > format short e; 1.350ge+Ol 1.6801e+03 1.0442e+06 1.039ge+09

for n == 3, 9, 15, 21, respectively. The lack of convergence is also indicated by the presence of severe oscilla- tions in the graph of the interpolating polynomial with respect to the graph of f, especially near the endpoints of the interval (see Figure 3.5, right). This behavior is known as Runge'8 phenomenon.

-2.7

2

-2.8 ... .

- - - "..,..."'..", ".'"" ".",., ,.J',.,.... - - -

". .....,..,..,. .....,. - -

1 . .....:....."....:..... - . - .

- - " ".. ....0'" '0" ,..."" "_,, '" '0" .., ." "" - . - . - -

-2.9 ....

".",....."., .,.....,. ....,.... - - - - - - - - . - -

"...i.....,., ,.:.. ,.. "..... ,. ..... . - - ,.,.T.". ,t...,., ..-,..... .....-. 1,., "
.... " - - - - . . - - - - - - - - - - . . . - - . - - -

- -- -1 ...,.....,...;.....,.....;.....

..... .:..... .....

- - -3 .. ..:..... ..:.. .. . . " .:.. .. .. - - . - - . - . - - - - - - - - . . - - -

-3.1 .. ..

.....:..... .....

..... -.....

.....:.. .....

- - -2 ... .;.....,

.....,.....

..... .:.....,..... ....

-3 .. ..

- - - .. ..., ...". ,.,..... ,.."... ...,.,.." .....,.,... ,.,....', "_'....," ".. It .

-3.4 .....

..... .

.. . .

... .

.. . ... - - - - - - - - - . - - - - -

-3.5 -60 -40 -20

o

20

40

60

-4 80 -5

-3

-1

3

5

Fig. 3.5. Two examples of Runge's phenomenon: to the left, II 12 f computed for the data of Table 3.1, column K == 0.67; to the right, II 12 f (solid line) computed on 13 equispaced nodes for the function f(x) == 1/(1 + x 2) (dashed line)

Remark 3.1 The following inequality can also be proved:

maxlf'(x) - (IInf)'(x)1 < Ch n ma x lf(n+l)(x)l, xEI xEI

where C is a constant independent of h. Therefore, if we approximate the first derivative of f by the first derivative of IInf, we loose an order of convergence with respect to h. In MATLAB, (IInf)' can be computed using the instruction [d] =polyder (c), where c is the input vector in which we store the coefficients of the interpolating polynomial, while d is the output vector where we store the coefficients of its first derivative (see Section 1.4.2).

See the Exercises 3.1-3.4.

... 'I'. '.. . 'Wjffi,

k

66 3. Approximation of functions and data

0.8

..

\ \ , :, , "' 3

/ r

/: I I ,

' I "I , I , I, ': \ I ,)' . " , I I , I , , "" f"," I 401' I'" , ... I " , · I,' I,'" I I I " ... '" I " ... .' I I I I f .....',\.,,'..." I I I

- -

- - - -.- - - _: 4': - - - -,- - - - 1. - _.! I I , I I I I , I I I I I I I I I I I I I I I I. I I I' .. . .

-1

5 -1 == Xo

.-...

o

X n == 1

-3

Xi

Fig. 3.6. The left side picture shows the comparison between the function f (x) == 1/(1 + x 2) (thin solid line) and its Chebyshev interpolating polynomi- als of degree 8 (dashed line) and 12 (solid line). Note that the amplitude of spurious oscillations decreases as the degree increases. The right side picture shows the distribution of Chebyshev nodes in the interval [-1, 1]

.

JJIJ

Runge's phenomenon can be avoided if a suitable distribution of nodes is used. In particular, in an arbitrary interval [a, b], we can consider the so called Chebyshev nodes (see Figure 3.6, right):

Xi==

a+b b-a.-... .-... (/) 2 + 2 Xi, where Xi = - cas 7ri n ,

i == 0, . . . , n.

Indeed, for this special distribution of nodes it is possible to prove that, if ! is a continuous and differentiable function in [a, b], IIn! converges to! as n

ex) for all x E [a,b]. The Chebyshev nodes, which are the abscissas of equispaced nodes on the unit semi-circle, lie inside [a, b] and are clustered near the endpoints of this interval (see Figure 3.6). Another non-uniform distribution of nodes in the interval [a, b], sharing the same convergence properties of Chebyshev nodes, is provided by:

x. = a + b _ b - a cas (2 i + 1 7r) i == 0 , . . . , n. '" 2 2 n+12'

Example 3.3 We consider anew the function f of Runge's example and com- pute its interpolating polynomial at Chebyshev nodes. The latter can be ob- tained through the following MATLAB instructions: » xc == -cos(pi*[O:n]/n); x == (a+b)*O.5+(b-a)*xc*O.5;

where n+1 is the number of nodes, while a and b are the endpoints of the interpolation interval (in the sequel we choose a=-5 and b=5). Then we compute the interpolating polynomial by the following instructions:

> > f== '1. / (1 + X. A 2) '; y == eva l (f); c == pol yfi t (x, y , n) ;

Now let us compute the absolute values of the differences between f and its Chebyshev interpolant at as many as 1001 equispaced points in the interval [-5,5] and take the maximum error values:

> > x == li n 5 pace (- 5,5, 1000); p == pol yv a l (c, x); fx == eva l (f); err == m a x (a bs (p- fx)) ;

As we see in Table 3.2, the maximum of the error decreases when n increases.

n 5 10 20 40 En 0.6386 0.1322 0.0177 0.0003

Tab. 3.2. The Chebyshev interpolation error for Runge's function f(x) == 1/(1 + x 2)

We want to approximate a periodic function f : [0,27r] -t __ JR., i.e. one satisfying f(O) == f(27r), by a trigonometric polynomial f which interpolates f at the n + 1 nodes Xj == 27rj /(n + 1), j == 0,..., n, i.e.

f(Xj) == f(xj), for j == 0,... ,n.

(3.8)

--- The trigonometric interpolant f is obtained by a linear combination of sines and cosines. In particular, if n is even, f will have the form

]\;1 -- ao """ f(x) = 2 +

[ak cos(kx) + b k sin(kx)] , k==l

(3.9)

where M == n/2 while, if n is odd,

M - ao """ f(x) = 2 +

[ak cos(kx) + b k sin(kx)] + aM+l cos((M + 1) x), (3.10) k==l

where M == (n - 1)/2. We can rewrite (3.9) as

M j(x)= L Ck eikx , k==-M

(3.11)

i being the imaginary unit. The coefficients Ck are related to the coeffi- cients ak and b k as follows:

ak == Ck + C-k, b k == i(Ck - C-k), k == 0,..., M. (3.12) Indeed, from (1.5) it follows that e ikx == cos(kx) + i sin(kx) and

M M L Ck eikx = L Ck (cos(kx) + i sin(kx)) k==-M k==-M M = L (ck(cos(kx) + isin(kx)) + c-k(cos(kx) - isin(kx))] + co. k==l

Therefore we derive (3.9), thanks to the relations (3.12). Analogously, when n is odd, (3.10) becomes

M+1 j(x) = L Ck eikx , k==-(M +1) where the coefficients Ck for k == 0, . . . , M are the same as before, while CM+1 == C-(M+1) == aM+1/2. In both cases, we could write

(3.13)

M +J-L j(x)= L Ck eikx , k==-(M+J-L) with J-L == 0 if n is even and J-L == 1 if n is odd. Because of its analogy with Fourier series, f is called a (real) discrete Fourier transform (DFT). Imposing the interpolation condition at the nodes Xj == jh, with h == 27r/(n + 1), we find that

(3.14)

M +J-L L ckeikjh = f(xj), j = 0, · · . ,n. k==-(M+J-L) For the computation of the coefficients {Ck} let us multiply equations (3.15) by e- imxj == e- imjh , wherem is an integer between 0 and n, and then sum with respect to j:

(3.15)

n M+J-L n L L ckeikjhe-imjh = Lf(xj)e- imjh . j==O k==-(M+J-L) j==O We now require the following identity:

(3.16)

n Leijh(k-m) = (n + l)<5 km , j==O

where 6km is the Kronecker symbol. This identity is obviously true if k == m. When k # m, we have n 1 _ (e i (k-m)h) n+l """" eijh(k-m) ==

1 - ei(k-m)h · j==O

The numerator on the right hand side is null, since 1 - e i (k-m)h(n+l) = 1 - e i (k-m)27r = 1 - cos((k - m)27r) - i sin((k - m)27r). Therefore, fro

(3.16) we get the following explicit expression for the coefficients of f:

n C = 1 """" f(x .)e -imjh m (M + II.) M + II. m n+1

J , ==- r--v'...' r--v j==O

The computation of all the coefficients C m can be accomplished with an order n log2 n operations by using the fast Fourier transform (FFT), which is implemented in the MATLAB program fft (see Example 3.4). fft Similar conclusions hold for the inverse transform through which we obtain the values {f (x j)} from the coefficients {Ck}. The inverse fast Fourier transform is implemented in the MATLAB program ifft. ifft

Example 3.4 Consider the function f(x) == x(x - 27r)e- X for x E [0, 27r]. To use the MATLAB program fft, we evaluate f at the nodes Xj == j7r /5 for j == 0, . . . , 9. Then, by the following instructions (and recalling that . * is the component-by-component vector product): » x==pi/5*[O:9]; y==x.*(x-2*pi).*exp(-x); Y==fft(y);

we compute

» Y == Columns 1 through 3 -6.5203e+OO -4.6728e-01 + 4.2001e+OOi 1.2681e+OO + 1.6211e+OOi Columns 4 through 6 1.0985e+OO + 6.0080e-Oli 9.2585e-01 + 2.1398e-01i 8.7010e-01 - 1.3887e-16i Columns 7 through 9 9.2585e-01 - 2.1398e-01i 1.0985e+OO - 6.0080e-01i 1.2681e+OO - 1.6211e+OOi Column 10 -4.6728e-01 - 4.2001e+OOi The coefficients {Ck} are simply given by Y / length (Y), where the command length computes the dimension of a vector (10 in this example). Note that the program ifft achieves the maximum efficiency when n is a power of 2, even though it works for any value of n.

interpft The command interpft provides the trigonometric interpolant of a set of data. It requires in input an integer N and a vector of values which represent the values taken by a function (periodic with period p) at the set of points Xi == iplIvf, i == 1,..., M - 1. interpft returns the N values of the trigonometric interpolant, obtained by the Fourier transform, at the nodes t i == iplN, i == 0,. . . , N - 1. For instance, let us reconsider the function of Example 3.4 in [0,27r] and

take its values at 10 equispaced nodes Xi == i7r 15, i == 0, . . . ,9. The values of the trigonometric interpolant at, say, the 100 equispaced nodes ti == i7r 1100, i == 0, . . . ,99 can be obtained as follows (see Figure 3.7) > > x==pi 15* [0:9]; y==x. * (x-2* pi). *exp(-x); z==i nterpft(y, 100);

0.5

-2.5 o

2

3

4

5

6

Fig. 3.7. The function f(x) == x(x - 27r)e- X (dashed line) and the corre- sponding trigonometric interpolant (continuous line) relative to 10 equispaced nodes

In some cases the accuracy of trigonometric interpolation can dramat- ically downgrade, as shown in the following example.

Example 3.5 Let us approximate the function f(x) == fl (x) + f2(X) where fl(X) == sin(x) and f2(X) == sin(5x), using nine equispaced nodes in the interval [0, 27r]. The result is shown in Figure 3.8. Note that in some intervals the trigonometric approximant shows even a phase inversion with respect to the function f.

This lack of accuracy can be explained as follows. At the nodes consid- ered, the function f2 is indistinguishable from f3(X) == -sin(3x) which has a lower frequency (see Figure 3.9). The function that is actually ap- proximated is therefore F(x) == fl(X) + f3(X) and not f(x) (in fact, the dashed line of Figure 3.8 does coincide with F).

-1

o

2

3

4

5

6

Fig. 3.8. The effects of aliasing: comparison between the function f(x) == sin(x) + sin(5x) (solid line) and its trigonometric interpolant (3.9) with M == 3 (dashed line)

This phenomenon is known as aliasing and may occur when the func- tion to be approximated is the sum of several components having differ- ent frequencies. As soon as the number of nodes is not enough to resolve the highest frequencies, the latter may interfere with the low frequen- cies, giving rise to inaccurate interpolants. To get a better approximation for functions with higher frequencies, one has to increase the number of interpolation nodes. A real life example of aliasing is provided by the apparent inversion of the sense of rotation of spoked wheels. Once a certain critical velocity is reached the human brain is no longer able to accurately sample the moving image and, consequently, produces distorted images.

1

, 'I \ . \'. I \ . , I \ \ :' I <D \ I

'. \ ., I \ , I . \ I : \ , \ , \ I \ , I '. I , . , \ , . I \ . I \ , I , , \ . , , , , \ , I , \ I \ , \ , , \ : I , \ , , , I I ,

, \ , \ , \ , I I , \ , I , I , , , \ \ - I I \ , , \ , , \ - I , \ I . \ I , I : : I .. , , , \ , , \ I , \ , I , , I

, -., \ , \ (!) \ : \ I I I , .., \ I: \ .' \ \ , \ I I.' I . . . . ,). \ 1 2 3 4 5 6

-0.5 -.,

-1 o

Fig. 3.9. The phenomenon of aliasing: the functions sin(5x) (dashed line) and - sin(3x) (dotted line) take the same values at the interpolation nodes. This circumstance explains the severe loss of accuracy shown in Figure 3.8

II

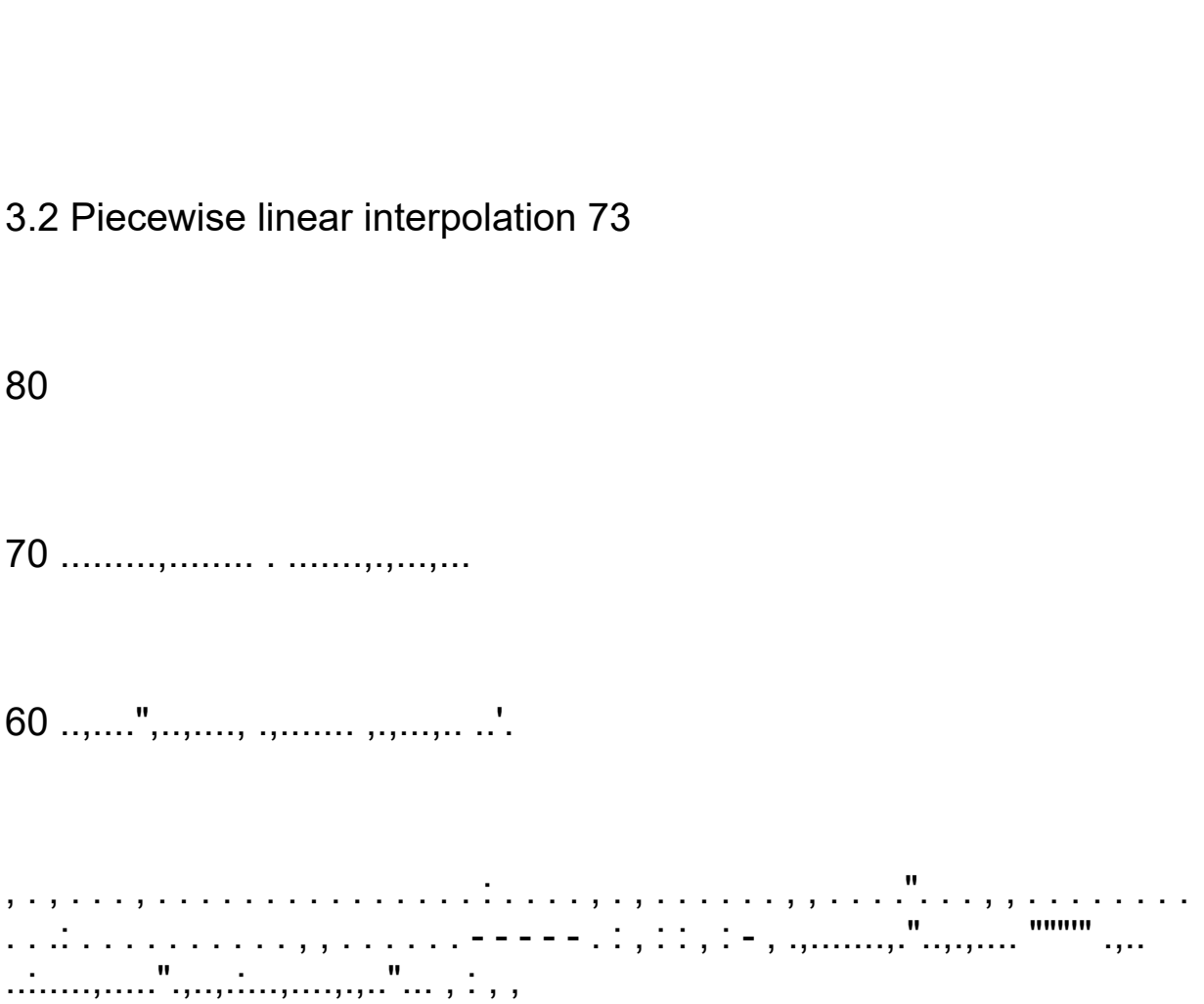
1. Approximating a set of data_ or a function f in [a, b] consists of finding a suitable function f that represents them with enough accuracy;

- 2. the interpolation process consists of determining a function f such that f (Xi) == Yi, where the {Xi} are given nodes and {Yi} are either the values {f(Xi)} or a set of prescribed values; 3. if the n+ 1 nodes {Xi} are distinct, there exists a unique polynomial of degree less than or equal to n interpolating a set of prescribed values {Yi} at the nodes {Xi}; 4. for an equispaced distribution of nodes in [a, b] the interpolation error at any point of [a, b] does not necessarily tend to 0 as n tends to infinity. However, there exist special distributions of nodes, for instance the Chebyshev nodes, for which this convergence property holds true for all continuous functions; 5. trigonometric interpolation is well

uited to approximate periodic functions, and is based on choosing f as a linear combination of sine and cosine functions. The FFT is a very efficient algorithm which allows the computation of the Fourier coefficients of a trigonometric interpolant from its node values and admits an equally fast inverse, the IFFT.

The Chebyshev interpolant provides an accurate approximation of smooth functions f whose expression is known. In the case when f is non smooth or when f is only known by its values at a set of given points (which do not coincide with the Chebyshev nodes), one can resort to a different interpolation method which is called linear composite interpolation. More precisely, given a set of nodes Xo < Xl < ... < X n , we denote by Ii the interval [Xi, Xi+l]. We approximate f by a continuous function which, on each interval, is given by the segment joining the two points (Xi, f(Xi)) and (Xi+l, f(Xi+l)) (see Figure 3.10). This function, denoted by rr{l f, is called a piecewise linear interpolation polynomial and its . . expreSSIon IS:

n{l f(x) = f(Xi) + f(Xi+l) - f(Xi) (x - Xi) for x E Ii. Xi+l - Xi


80
70
60
o -2
o
2
4

6

8

Fig. 3.10. The function f(x) == x 2 + 10j(sin(x) + 1.2) (solid line) and its piecewise linear interpolation polynomial IT {I f (dashed line)

The upper-index H denotes the maximum length of the intervals Ii' The following result can be inferred from (3.7) setting n == 1 and h == H:

Proposition 3.3 If f E C 2 (I), where I == [xo, x n], then

H 2 max\f(x) - IIf f(x)1 < -maxlf"(x)\. xEI 8 xEI

Consequently, for all x in the interpolation interval, II {I f (x) tends to f(x) when H ---+ 0, provided that f is sufficiently smooth. Through the instruction sl=interpl (x, y, z) one can compute the interpl values at arbitrary points, which are stored in the vector z, of the piece- wise linear polynomial that interpolates the values y (i) at the nodes x(i), for i = 1,... ,n+l. Note that z can have arbitrary dimension. If the nodes are in increasing order (i.e. x(i+l) > xCi), for i=l,... ,n) then we can use the quicker version interplq (q stands for quickly). interplq It is worth mentioning that the command fplot, which is used to display the graph of a function f on a given interval [a, b], does in- deed replace the function by its piecewise linear interpolant. The set of interpolating nodes is generated automatically from the function,

follow- ing the criterion of clustering these nodes around points where f shows strong variations. A procedure of this type is called adaptive.

" ... . . :..... .:j::):<: ... .,..

The main drawback of piecewise linear interpolation is that rrf f is noth- ing more than a global continuous function. As a matter of fact, in several applications, e.g. in computer graphics, it is desirable to get approxima- tion by smooth functions which have at least a continuous derivative. With this aim, we can construct a function 83 with the following prop- erties:

1. on each interval Ii == [Xi, Xi+l], for i == 0, . . . , n - 1 83 is a polyno- mial of degree 3 which interpolates the pairs of values (Xj, f(xj));

2. 83 has continuous first and second derivatives on [xo, x n].

For its complete determination, we need 4 conditions on each inter- val, therefore a total of 4n equations, which we can provide as follows: n + 1 conditions arise from the interpolation requirement at the nodes Xj; requiring the continuity of the polynomial at the internal nodes Xl, . · · , Xn-l yields n -1 further equations; we obtain 2(n -1) newequa- tions by requiring that both first and second derivatives be

continuous at the internal nodes. Finally, we still lack two further equations, which we can e.g. choose as

8

(Xo) == 0, 8

(Xn) == o.

(3.1 7)

The function 83 which we obtain in this way, is called a natural interpolating cubic spline. By choosing suitably the unknowns (see [QSSOO], Section 8.6.1) to represent 83 we arrive at a (n + 1) x (n + 1) system with a tridiagonal matrix whose solution can be accomplished by a number of operations proportional to n (see Section 5.4). The choice (3.17) is not the only one possible to complete the system of equations. Several alternatives do exist. One possibility would be to replace (3.17) by requiring the continuity of the third derivative of 83 at the nodes Xl and Xn-l. This is precisely what we get when using the spline MATLAB command spline (see also the toolbox splines). The input parameters are the vectors x and y, as well as the vector z containing the points at which we are seeking the values of 83.

Example 3.6 Let us reconsider the data of Table 3.1 corresponding to the column K == 0.67 and compute the associated cubic spline 83. The different values of the latitude provide the nodes Xi, i == 0, . . . , 12. If we are interested in computing the values 83(Zi), where Zi == -55 + i, i == 0,. . . , 120, we can proceed as follows:

> > x == [-55: 10:65]; > > y == [-3.25 -3.37 -3.35 -3.2 -3.12 -3.02 -3.02... -3.07 -3.17 -3.32 -3.3 -3.22 -3.1]; » z == [-55:1:65]; > > 5 == spline(x,y,z);

The graph of 83, which is reported in Figure 3.11, looks more plausible than that of the Lagrange interpolant at the same nodes.

-2.7

- - - . . . - ""....,.. ",.,....".... " . ".,..,....". ",... ",..... ",....., . ",. . . . .

- , : -3.4 ..... -. . -,. ..:..,. ... .:.. ... .:. .., ...

. . . .:. . . ..:. ... .. , :' : : - : : \ :I - :

:

-3.5 -60

-40

-20

o

20

40

60

Fig. 3.11. Comparison between the cubic spline and the Lagrange interpolant for the case considered in Example 3.6

The error that we obtain in approximating a function f (continuously differentiable up to its fourth derivative) by the natural cubic spline satisfies the following inequalities:

maxlf(r) (x) - s

r)(x)1 < CrH4-rmaxlf(4) (x)l, r = 0,1,2,3, xEI xEI

where I == [xo, xn] and H == maxi==O,...,n-l (Xi+l - Xi), while C r is a suitable constant depending on r, but independent of H. It is then clear that not only f, but also its first, second and third derivatives are well approximated by 83 when H tends to o.

Remark 3.2 Cubic splines in general don't preserve monotonicity between neighbouring nodes. For instance, by approximating the unitary circumference in the first quarter using the points (Xk == sin(k7r/6),Yk == cos(k7r/6)), for k == 0, . . . ,3, we would obtain an oscillatory spline (see Figure 3.12). In these cases, other approximation techniques can be better suited. For instance, the MATLAB command pchip provides the Hermite piecewise cubic interpolant pchip and guarantees the local monotonicity of the interpolant (see Figure 3.12). The Hermite interpolant can be obtained by using the following instructions:

> > t == linspace(O,pij2,4) > > x == cos(t); y == si n (t) ; » xx == linspace(O,1,40); > > plot(x,y, 's' ,xx, [pchip(x,y,xx) ;spline(x,y,xx)])

0.8 .. ..

- - : : -- -- ....: ----'*: ..

,

.,

.

,.....:.,...,.....

, , : .... "," ,.':'",. , : , : \ : \: . \ 0.6 ",.,.."""", '

' , , , , " , " , , " , '

' . , " .; ,... , , , , -" " ., '" """; \, , ,

0.6

0.4 .. ".. """"-" ""...,.. "-""""",,, ""..... "". ..,..... """ ,

0.2 """""""""":""""""""":""""""

- - . - . , I , ..., ... , .,. ,. ., , , I. ,.., ., ., . -

o o

0.2

0.4

0.8

Fig. 3.12. Approximation of the first quarter of the circumference of the unitary circle using only 4 nodes. The dashed line is the cubic spline, whereas the continuous line is the piecewise cubic Hermite interpolant

.. .. . , ...-. .' %K

lW>

See the Exercises 3.5-3.8.

We have already noticed that a Lagrange interpolation does not guar- antee a better approximation of a given function when the polynomial degree gets large. This problem can be overcome by composite interpo- lation (such as piecewise linear polynomials or splines). However, neither are suitable to extrapolate information from the available data, that is, to generate new values at points lying outside the interval where inter- polation nodes are given.

Example 3.7 On the basis of the data reported in Figure 3.1, we would like to predict whether the stock price will increase or diminish in the coming days. The Lagrange polynomial interpolation is impractical, as it would re- quire a (tremendollsly oscillatory) polynomial of degree 719 which will provide a completely useless

prediction. On the other hand, piecewise linear interpo- lation, whose graph is reported in Figure 3.1, provides extrapolated results

by exploiting only the values of the last two days, thus completely neglecting the previous history. To get a better result we should avoid the interpolation requirement, by invokillg least square approximation as indicated below.

Assume that th

data {(Xi, f(Xi)), i == 0, . . . , n} are available. We look for a polynomial f of degree at most m > 1 which satisfies the following inequality:

n

n

""""" - 2 """""" 2 L.)!(Xi) - !(Xi)] < L.)!(Xi) - Pm (Xi)] i==O i==O

(3.18)

-- for every polynomial Pm of degree at most m. Should it exist, f will be called the least square approximation of f. For arbitrary values of m and n, it will not be possible to guarantee that f(Xi) == f(Xi) for all i == 0, . . . , n. Setting

f(x) == ao + aIX + . . . + amx m ,

(3.19)

where the coefficients ao, . . . , am are unknown, the problem (3.18) can be restated as follows:

<I>(aO, aI,. · · , am) == min <I>(b o , b I ,. · · , b m) { bi, i==O,..., m }

where

n <p(b o , b 1 ,. · · , b m) = L [J(Xi) - (b o + b1Xi + . . . + bmX

)]2 · i==O

Let us solve this problem in the special case when m == 1. Since

n <p(b o , bd = L [!(Xi)2 + b6 + bix; + 2b o b 1 X i - 2b o !(Xi) - 2b 1 X i!(Xi)] , i==O

the graph of <I> is a convex paraboloid. The point (ao, aI) at which <I> attains its minimum satisfies the conditions:

8 <I> Db o (ao, al) = 0,

8 <I> Db 1 (ao,al) = 0,

where the symbol 8<P /8b j denotes the partial derivative (that is, the rate of variation) of <P with respect to b j , after having frozen the remaining variable (see the definition 8.3).

By explicitly computing the two partial derivatives we obtain

n

n 2:)aoxi + alX; - Xi!(Xi)] = 0, i==O

L[ao + alXi - !(Xi)] = 0, i==O

which is a system of 2 equations for the 2 unknowns ao and al:

n n

ao(n + 1) + al LXi = L!(Xi), i==O i==O n n n

(3.20)

aOLXi + alL x ; = L!(Xi)Xi. i==O i==O i==O

Setting D == (n + 1) 2:

0 x'f - (2:

0 Xi)2, the solution reads:

1 ao == - D

n

n

n

n

L!(Xi) LX; - LXj LXi!(Xi) , i==O j ==0 j ==0 i==O

1 al == - D

n

n

n

n

(3.21)

(n + 1) LXi!(Xi) - LXj L!(Xi) · i==O j==O i==O

'" The corresponding polynomial f is known as the least square straight line, or regression line. The previous approach can be generalized to arbitrary m. The associ- ated (m + 1) x (m + 1) linear system, which is symmetric, will have the following form:

n n n ao(n + 1) +alLxi + · · · + am L x

= L!(Xi), i==O i==O i==O n n n n ao LXi +alL x ; + L m+l = LXi!(Xi), + · · . am Xi i==O i==O i==O i==O

n aoLx

i==O

n +al LX

+l i==O

n + · · · + am L x;m i==O

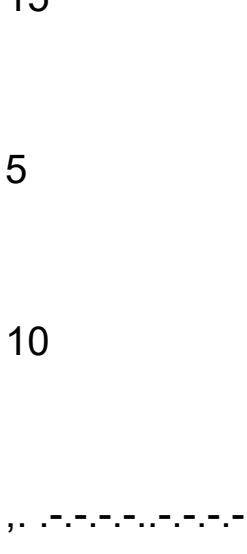
n

= LX

!(Xi). i==O

When m == n, the least square polynomial must coincide with the Lagrange interpolating polynomial IInf (see Exercise 3.9). The MATLAB command c=polyf i t (x , Y , m) computes by default the coefficients of the polynomial of degree m which approximates n+l pairs of data (x (i) , y (i)) in the least square sense. As already noticed in Section 3.1.1, when m is equal to n it returns the interpolating polynomial.

Example 3.8 In Figure 3.13 we draw the graphs of the least square poly- nomials of degree 1, 2 and 4 that approximate the data of Figure 3.1. The polynomial of degree 4 reproduces quite reasonably the behavior of the stock price in the considered time interval and suggests that in the near future the quotation will increase.



novOO

may01

nov01

may02

Fig. 3.13. Least squares approximation of the data of Problem 3.2 of degree 1 (dashed-dotted line), degree 2 (dashed line) and degree 4 (thick solid line). The exact data are represented by the thin solid line

.....w""

.::...

1. The composite piecewise linear interpolant of a function f is a piecewise continuous linear function f, which interpolates f at a given set of nodes {Xi}. With this approximation we avoid Runge's type phenomena when the number of nodes increases;

2. interpolation by cubic splines allows the approximation of f by a piecewise cubic function f which is continuous together with its first

and second derivatives;

3. in least squares approximation we look for an approximant f which is a polynomial of degree m (typically, m < n) that minimizes the mean-square error L:

0 [j (Xi) - j (Xi) j2 ·

See the Exercises 3.9-3.13.

. .

. .:.--: .

t::'

F . ,. . ..." .... :.:. . ::.....:.:.:.:.:..-:.:-:-:.:. . .. . :. . . ..

For a more general introduction to the theory of interpolation and approximation the reader is referred to [Dav63], [Mei67] and [Gau97]. Polynomial interpolation can also be used to approximate data and functions in several dimensions. In particular, composite interpolation, based on piecewise linear or spline functions, is well suited when the region n at hand is partitioned into polygons in 2D (triangles or quadri- laterals) and polyhedra in 3D (tetrahedra or prisms). A special situation occurs when n is a rectangle or a

parallelepiped. In that case one could simply use the commands zi=interp2 (x, y , z, xi , yi) or vi=interp3(x,y,z,v,xi,yi,zi), respectively, x, ..., v being the interpolating values and xi , ..., zi the nodes at which the inter- polating polynomial should be evaluated. For instance, to approximate by a cubic spline the values of the func- tion f(x, y) == sin(21rx) cos(21rY) on a uniform grid of 36 nodes on the square [0,1]2, we use the following instructions: > > [x,y] ==meshgrid (0:0.2: 1 ,0:0.2: 1); z==si n (2* pi*x). *cos(2* pi *y); Then, the cubic interpolating spline, evaluated on a uniform grid of 441 nodes (21 in both x and y directions), can be obtained as follows: » xi == [0:0.05:1]; yi==[0:0.05:1]; > > [xf,yf]==meshgrid (0:0.05: 1,0:0.05: 1); pi3==i nterp2(x,y,z,xf,yf,' spl i ne'); meshgrid The command meshgrid transforms the domain specified by the vectors x and y into arrays xf and yf that can be used for the evaluation of a function of two variables and for 3-dimensional MATLAB surface plots. The rows of the matrix xf are copies of the vector xi and the columns of the matrix yf are copies of the vector yi. Alternatively, one can use the griddata function griddata (also available for 3-dimensional data (griddata3) or for n-dimensional hypersurface fitting (griddatan). When n is a two-dimensional domain of arbitrary shape, it can be pdetool partitioned into triangles using the graphical interface pdetool. For a general presentation of spline functions see, e.g., [Die93] and [PBP02]. The MATLAB toolbox splines allows one to explore several spdemos applications of spline functions. In particular, the spdemos command gives the user the possibility to investigate the properties of the most important type of spline functions. Rational splines, i. e. functions which are the ratio of two splines functions, are accessible through the com- rpmak mands rpmak and rsmak. Special instances are the so-called NURBS rsmak splines, which are commonly used in CAGD (Computer Assisted Geo- metric Design).

interp2 interp3

In the same context of Fourier approximation, we mention the approx- imation based on wavelets. This type of approximation is largely used for image reconstruction and compression and in signal analysis (for an introduction, see [DL92], [Urb02]). A rich family of wavelets (and their applications) can be found in the MATLAB toolbox wavelet. wavele1

Exercise 3.1 Prove inequality (3.6).

Exercise 3.2 Provide an upper bound of the Lagrange interpolation error for the following functions:

fl(X)

cosh(x), f2(X)

sinh(x), f3(X)

cos(x) + sin(x),

Xk

-1 + 0.5k, k

0, . . . ,4, xk

-7r/2+7rk/4, k

0,...,4.

Exercise 3.3 The following data are related to the life expectation of citizens of two European regions:

	Western Europe	Eastern Europe
1975	72.8	70.2
1980	74.2	70.2
1985	75.2	70.3
1990	76.4	71.2

Use the interpolating polynomial of degree 3 to estimate the life expectation in 1970, 1983 and 1988. Then extrapolate a value for the year 1995. It is known that the life expectation in 1970 was 71.8 years for the citizens of the West Europe, and 69.6 for those of the East Europe. Recalling these data, is it possible to estimate the accuracy of life expectation predicted in the 1995?

Exercise 3.4 The price (in euros) of a magazine has changed as follows:

N ov.87 4.5

Dec.88 5.0

N ov. 90 6.0

Jan.93 6.5

Jan.95 7.0

Jan.96 7.5

N ov. 96 8.0

N ov. 00 8.0

Estimate the price in November 2002 by extrapolating these data.

Exercise 3.5 Repeat the computations carried out in Exercise 3.3, using now the cubic interpolating spline computed by the function

spline. Then compare the results obtained with the two approaches.

Exercise 3.6 In the table below we report the values of the sea water density p (in Kg/m 3) corresponding to different values of the temperature T (in degrees Celsi us) :

T 4° 8° p 1000.7794 1000.6427

12° 1000.2805

16° 999.7165

20° 998.9700

Compute the associated cubic interpolating spline on 4 subintervals of the temperature interval [4, 20]. Then compare the results provided by the spline interpolant with the following ones (which correspond to further values of T):

T 6° p 1000.74088

10° 1000.4882

14° 1000.0224

18° 999.3650

Exercise 3. 7 The Italian production of citrus fruit has changed as follows:

year production (x 10 Kg)

1965 17769

1970 24001

1980 25961

1985 34336

1990 29036

1991 33417

Use a cubic interpolating spline to estimate the production in 1962, 1977 and 1992. Compare these results with the real values: 12380, 27403 and 32059, respectively. Repeat the computation using the Lagrange interpolation poly- nomial.

Exercise 3.8 Evaluate the function f(x) == sin(27rx) at 21 equispaced nodes in the interval [-1,1]. Compute the Lagrange interpolating polynomial and the cubic interpolating spline. Compare the graphs of these two functions with that of f on the given interval. Repeat the same calculation using the following perturbed set of data: f(Xi) == (_1)i+l10- 4 , and observe that the Lagrange interpolating polynomial is more sensitive to small perturbations than the cubic spline.

Exercise 3.9 Verify that if m == n the least-square polynomial of a function f at the nodes XO, . . . , X n coincides with the interpolating polynomial IInf at the same nodes.

Exercise 3.10 Compute the least-squares polynomial of degree 4 that ap- proximates the data of the second, third and fourth columns of Table 3.1.

Exercise 3.11 Repeat the computations carried out in Exercise 3.7 using now a least-squares approximation of degree 3.

Exercise 3.12 Express the coefficients of system (3.20) in terms of the aver- age M = (N

l) 2:

0 Xi and the variance v = (N

1) 2:

O(Xi - M)2 of the set of data {Xi, i == 0, . . . , n}.

Exercise 3.13 Verify that the regression line passes through the point whose abscissa is the average of {Xi} and ordinate is the average of {f (Xi) }.

4. Numerical differentiation and integration

In this chapter we propose methods for the numerical approximation of derivatives and integrals of functions. Concerning integration, it is known that for a generic function it is not always possible to find a primitive in an explicit form. In other cases, it could be hard to evaluate a primitive as, for instance, in the example

l J 1r . (3) 4 00 (-9/4)k 7r cos(4x)cos(3sm(x)) dx = 2 2: k !(k+4)! ' o k

O

where the problem of computing an integral is transformed into the equally troublesome one of summing a series. It is also worth

mention- ing that sometimes the function that we want to integrate or differentiate could only be known on a set of nodes (for instance, when the latter rep- resent the results of an experimental measurement), exactly as happens in the case of the approximation of functions, which was discussed in chapter 3. In all these situations it is necessary to consider numerical methods in order to obtain an approximate value of the quantity of interest, in- dependently of how difficult is the function to integrate or differentiate.

Problem 4.1 (Hydraulics) The height q(t) (in meters) of a flow in a conical funnel with a circular hole of radius r == 10 cm is measured at different time intervals, yielding the following values:

t q(t)

0.0 1.0

5 0.8811

10 0.7366

15 0.5430

20 0.1698

We want to compute an approximation of the emptying veloc ity q' (t) and compare it with the analytical one, given by q' (t) == -0.61rr 2 V 19.6q(t) / A(t)

(A(t) is the area of the horizontal section of the cone corresponding to a fluid height q(t)). For the solution of this problem, see Example 4.1. .

Problem 4.2 (Optics) In order to plan a room for infrared beams we are interested in calculating the energy emitted by a black body (that is, an object capable of irradiating in all the spectrum to the ambient ten1perature) in the (infrared) spectrum comprised between 3JLm and 14JLm wavelength. The solution of this problem is obtained by computing the integral

E(T) == 2.39 . 10- 11

14.10- 4 J

dx x5(el.432/CTx) - 1)'

(4.1)

3.10- 4

which is the Planck equation for the energy E(T), where x is the wavelength (in cm) and T the temperature (in Kelvin) of the black body. For the computation of the above integral, see Exercise 4.17. .

Problem 4.3 (Electromagnetism) Consider an electric wire sphere of ar- bitrary radius r and conductivity a. We want to compute the density distri- bution of the current j as a function of rand t (the time), knowing the initial distribution of the current density p(r). The problem can be solved using the relations between the current density, the electric field and the charge density and observing that, for the symmetry of the problem, j(r, t) == j(r, t)r/lr/, where j == UI. We obtain

j(r, t) == j(r)e- at / EO ,

T j(r) = (j 2 J p(

)

2 d

, sor o

(4.2)

where So == 8.859 . 10- 12 farad/m is the dielectric constant of the void. For the computation of j(r), see Exercise 4.16. .

Consider a function f : [a, b] --+

continuously differentiable in [a, b] . We seek an approximation of the first derivative of f at a generic point x in (a, b) . In view of the definition (1.9), for h sufficiently small and positive, we can assume that the quantity

(<5+f)(x) = f(x + h) - f(x) h

(4.3)

is an approximation of f'(x) which is called the forward finite difference. To estimate the error, it suffices to expand f in a Taylor series, obtaining h 2 f(x + h) = f(x) + hf'(x) + 2 f"(

)

(4.4)

where

is a suitable point in the interval (x, x + h). Therefore

(o+f)(x) = f'(x) +

f"(

),

(4.5)

and thus (6+f) (x) provides a first-order approximation to f' (x) with respect to h. With a similar procedure, we can derive from the Taylor . expanSIon

h 2 f(x - h) = f(x) - hf'(x) + 2 f"(1]) with 7] E (x - h, x), the backward finite difference

(4.6)

(o-f)(x) = f(x) -

(x - h)

(4.7)

which is also first-order accurate. Note that formulae (4.3) and (4.7) can also be obtained by differentiating the linear polynomial interpolating f at the points {x, x + h} and {x - h,x}, respectively. In fact, from the geometrical viewpoint, these schemes amount to approximating f'(x) by the slope of the straight line passing through the two points (x, f (x)) and (x + h, f(x + h)), or (x - h, f(x - h)) and (x, f(x)), respectively (see Figure 4.1). Finally, we introduce the centered finite difference formula

(of)(x) = f(x + h) - f(x - h) 2h

(4.8)

which provides a second-order approximation to f'(x) with respect to h. Indeed, we obtain

f'(x) - (of)(x) = h 2 [J"'(

) + f"'(1])], 12

(4.9)

where 7] and

are suitable points in the intervals (x - h, x) and (x, x + h), respectively (see Exercise 4.2). By (4.8) f'(x) is approximated by the slope of the straight line passing through the points (x - h, f(x - h)) and (x + h, f(x + h)).

-

ml == (b-f)(x)

m2 == (b+f)(x)....

-

-

-

-

, I - m3:= (8f)(x) , I I I I I I I I I I .

..

x-h

- X

x+h

Fig. 4.1. Finite difference approximation of f' (x): backward (solid line), for- ward (dotted line) and centered (dashed line). m 1, m2 and m3 denote the slopes of the three straight lines

Example 4.1 Let us solve Problem 4.1, using formulae (4.3), (4.7) and (4.8) to approximate q' (t). We obtain:

t 0.0 5 10 15 20 q' (t) -0.0220 -0.0259 -0.0326 -0.0473 -0.1504 6+q -0.0238 -0.0289 -0.0387 -0.0746 -- 6-q -- -0.0238 -0.0289 -0.0387 -0.0746 6q -- -0.0263 -0.0338 -0.0567 --

The agreement between the exact derivative and the one computed from the finite difference formulae with h == 5 is more satisfactory when using formula (4.8) rather than (4.7) or (4.3).

In general, we can assume that the values of f are available at n + 1 equispaced points Xi == Xo + ih, with h > o. In this case in the numerical derivation f'(Xi) can be approximated by taking one of the previous formulae (4.3), (4.7) or (4.8) with x == Xi. Note that the centered formula (4.8) can be used only at the internal points Xl,.. h' Xn-l but not at the extrema Xo and Xn. At the latter nodes we could use the values 1 2h [-3f(xo) + 4f(xd - f(X2)] at Xo, 1 2h [3f(xn) - 4f(xn

d + f(X n -2)] at X n ,

(4.10)

.:....tl #W;b., ::::::::::;::::::"

:::::<::::::::::::

which are also second-order accurate with respect to h. They are obtained by computing at the point Xo (respectively, x n) the first deriva- tive of the polynomial of degree 2 interpolating f at the nodes Xo, Xl, X2 (respectively, X n -2, Xn-l, x n).

See the Exercises 4.1-4.4.

In this section we introduce numerical methods suitable for approximat- ing the integral

b I(f) = J f(x) dx,

a

where f is an arbitrary continuous function in [a, b]. We start by introducing some simple formulae, which are indeed special instances of the broader family of Newton-Cotes formulae.

.

A simple procedure to approximate 1(f) can be devised by partitioning the interval [a, b] into subintervals 1k == [Xk-l, Xk], k == 1,..., M, with Xk == a + kH, k == 0,..., M and H == (b - a)/M. Since

M I(f) = L J f(x) dx, k==llk

(4.11)

on each sub-interval 1k we can approximate the exact integral of f by that of a polynomial f approximating f on 1k. The simplest solution consists of choosing f as the constant polynomial interpolating f at the middle point of I k :

- Xk ==

Xk-l + Xk 2

In such a way we obtain the composite midpoint quadrature formula

M l

p(f) = HLf(Xk) k==l

(4.12)

The symbol mp stands for midpoint, while c stands for composite. This formula is second-order accurate with respect to h. More precisely, if f is continuously differentiable up to its second derivative, we have

l(f) - l

p(f) = b

a H 2 f" (

),

(4.13)

where

is a suitable point in (a, b) (see Exercise 4.6). Formula (4.12) is also called the composite rectangle quadrature formula because of its geometrical interpretation, which is evident from Figure 4.2.

x

x

- Xo

Xk

- XM

a

(a + b)/2

b

Fig. 4.2. The composite midpoint formula (left); the midpoint formula (right)

The classical midpoint formula (or rectangle formula) is obtained by taking M == 1 in (4.12), i.e. using the midpoint rule directly on the interval (a, b):

Imp(f) == (b - a)f((a + b)/2]

The error is IIOW given by

1(f) - 1mp(f) = (b ;4 a)3 f"(

)

(4.14)

where

is a suitable point in (a, b). Relation (4.14) follows as a special case of (4.13), but it can also be proved directly. Indeed, setting x == (a + b)/2, we have

b 1(f) - 1mp(f) =! [J(x) - f(x)] dx

a b b = ! f' (x)(x - x) dx +

! f" (ry(x))(x - X)2 dx,

a

a

where TJ(X) is a suitable point in the interval whose endpoints are x and x. Then (4.14) follows because J: (x - x) dx = 0 and, by the mean value theorem, 3

E (a, b) such that

b b

! f"(ry(x))(x - X)2 dx =

f"(

) !(x - X)2 dx = (b ;4 a) 3 f"(

). a a

The degree of exactness of a quadrature formula is the maximum inte- ger r > 0 for which the approximate integral (produced by the quadra- ture formula) of any polynomial of degree r is equal to the exact integral. Thus, the midpoint formula has degree of exactness 1, since it integrates exactly all polynomials of degree less than or equal to 1 (but not all those of degree 2). The Inidpoint composite quadrature formula is ilnplemented in Pro- gram 3. Input parameters are the endpoints of the integration interval a and b, the number of subintervals M and a string f to define the function f. In particular, in order to ensure a correct execution of the program if f is a constant function, f must be provided in the form ' c+O . *x ' , where c is the constant value of f.

function 1m p==mid pointc(a, b, M ,f) %MIDPOINTC Composite midpoint numerical integration. % IMP == MIDPOINTC(A,B,M,FUN) computes an approximation of the integral % of the function FUN via the midpoint method (with M equispaced intervals). % FUN accepts real scalar input x and returns a real scalar value. % FUN can also

be an in line object. H==(b-a) /M; x == linspace(a+H/2,b-H/2,M); fmp==feval(f,x); Imp==H*sum(fmp);

See the Exercises 4.5-4.8.

.

Another formula can be obtained by replacing f on Ik by the linear poly- nomial interpolating f at the nodes Xk-1 and Xk (equivalently, replacing f by rr{l f, see Section 3.2, on the whole interval (a, b)). This yields

If(f)

H M = 2 L[J(Xk) + f(Xk-d] k==l H 1\1/-1 = 2[J(a) + f(b)] + H L f(Xk) k==l

(4.15)

This formula is called the composite trapezoidal formula, and is second- order accurate with respect to H.

1",

... . .. >Ji;:t: j ; .:> . . .)

- .

'... i

::_ ....,.. . .. .

In fact, one can obtain the expression

l(f) - 1

(f) = - b ;2 a H 2 f" (

)

(4.16)

for the quadrature error for a suitable point

E (a, b), provided that f E C 2 (a, b). When (4.15) is used with M == 1, we obtain

b-a 1t(f) = 2 [J(a) + f(b)]

(4.17)

which is called the trapezoidal formula because of its geometrical inter- pretation. The error induced is given by

l(f) - 1t(f) = - (b

2a)3 f"(

),

(4.18)

where

is a suitable point in (a, b), from which we can deduce that (4.1 7) has degree of exactness equal to 1, as is the case of the midpoint rule. With a simple modification of this procedure we can obtain a more accurate formula. In fact, we can still approximate f by a polynomial, but this time we consider as interpolatory points the Gauss nodes

Xk-l + Xk 1 (Xk - Xk-l) (1 1) H rk-l == - - == Xk-l + - - - 2 J3 2 J3 2' _ Xk-l + Xk

(Xk - Xk-l) _ (

) H rk - + ;;:; - Xk-l + 1 +;;:; · 2 v3 2 v3 2

With this choice we obtain the Gauss quadrature formula

H M 1causs(f) = "2 L [J('"Yk-d + f('"Yk)] k==O

(4.1 g)

Xo == a

Xk

X XM == b

Xo == a

X Xl == b

Fig. 4.3. Composite trapezoidal formula (left); trapezoidal formula (right)

Its order of accuracy (with respect to H) is equal to 4. Precisely,

l(f) - 1causs(f) =

2

H4f(4)(

)

where

is a suitable point in (a, b) (see [RR85]). Restricted to only one interval, (4.19) becomes:

(b - a) 1causs(f) = 2 (f(,I) + f(,o)]

(4.20)

where

_ a+b _

(b-a) ,0 - 2 J3 2 '

= a+b

(b-a) 'I 2 + J3 2 ·

The corresponding error is

_ (b - a)5 (4) I(f) - 1causs(f) - 4320 f (17),

for a suitable TJ E (a, b). In particular, it follows that this formula has degree of exactness equal to 3, i.e. it computes exactly the integral of all polynomials of degree less than or equal to 3 (but not those of degree 4). The quadrature formulae (4.15) and (4.19) are implemented in Pro- gram 4. The selection criterion is to set choice=1 for the trapezoidal rule, choice=2 for the Gauss rule. The trapezoidal formula (4.17) is im- plemented in the MATLAB program trapz, the composite formula (4.15) in cumtrapz.

function [Itpc]==trapc(a, b, M, f,choice, varargin) % TRAPC Composite two-points numerical integration. % IT PC == TRAPC(A,B,M,FUN,CHOICE) computes an approximation of the % integral of the function FUN via the trapezoidal method (using M equispaced % intervals) if CHOICE==I, via the Gauss composite formula if CHOICE==2. % FUN accepts real scalar input x and returns a real scalar value. % FUN can also be an inline object. H== (b-a) 1M; switch choice case 1, x==linspace(a, b, M+ 1); fpm==feval(f,x,varargin{:}); fpm (2 :end-I)==2*fpm (2 :end-I);

trapz cumtrapz

I;,

Itpc==O.5* H *su m(fpm); case 2, z==linspace(a, b, M+ 1); x==z(1 :end-l)+H /2*(1-1/ sq rt(3)); x==[x, x+H/sqrt(3)]; fpm==feval(f,x,varargin{:}); Itpc==O.5*H*sum(fpm); otherwise disp('Unknown formula'); end

Example 4.2 We want to compare the approximations of the integral l(f) == J;7r xe- x cos(2x) dx = -0.122122604618968, obtained by using the composite lllidpoint, trapezoidal and Gauss formulae. In Figure 4.4 we plot on the loga- rithmic scale the errors that are obtained versus H. As noticed in Section 1.5, in this type of plot the greater the slope of the curve, the higher the order of convergence of the corresponding formula. As expected from the theoreti- cal results, the midpoint and trapezoidal formulae are secol1d-order accurate, whereas the Gauss formula is fourth-order accurate.

10° 10- 2 10- 4 ,; 10- 6 ,

2 10- 8 10- 10 4 10- 12 10- 14 10- 3 10- 2 10- 1 10° 10 1

Fig. 4.4. Logarithmic representation of the errors versus H for Gauss (solid line with circles), midpoint (solid line) and trapezoidal (dashed

line) composite quadrature formulae

t

. ...

:.:;.

. . ,".-." - . ...

: ::.':-". . . . ':':':':

::.';....':'., .:. ,"

See the Exercises 4.9-4.11.

The Simpson formula can be obtained by replacing the integral of f over each I k by that of its interpolating polynomial of degree 2 at the nodes

Xk-l, Xk == (Xk-l + xk)/2 and Xk,

( ) _ 2(x - Xk)(X - xk) f( ) II 2 f x - H2 Xk-l 4(Xk-l - x)(x - Xk) f(-) 2(x - Xk)(X - Xk-l) f( ) + H2 x k + H2 x k ·

The resulting formula is called the Simpson composite quadrature for- mula, and reads

H M 1

(f) = "6 L [J(xk-d + 4f(Xk) + f(Xk)] k==l

(4.21)

One can prove that it induces the error

l(f) - 1

(f) = - \

oa

: f(4)(

),

(4.22)

where

is suitable point in (a, b), provided that f E 0 4 (a, b). It is there- fore fourth-order accurate with respect to H. When (4.21) is applied to only one interval, say (a, b), we obtain the so-called Simpson quadrature formula

b-a 1 8 (f) == [f(a) + 4f((a + b)/2) + f(b)J 6

(4.23)

The error now is given by

l(f) - 1 8 (f) = - 11 6 (b

;)5 f(4)(

),

(4.24)

for a suitable

E (a, b). Its degree of exactness is therefore equal to 3. The composite Simpson rule is implemented in Program 5.

function [Isic]==simpsonc(a, b, M, f, varargin) %SIMPSONC Composite Simpson numerical integration. % ISIC == SIMPSONC(A,B,M,FUN) computes an approximation of the integral % of the function FUN via the Simpson method (using M equispaced intervals). % FUN accepts real scalar input x and returns a real scalar value. % FUN can also be an inline object.

H==(b-a)/M; x==linspace(a, b, M+ 1); fpm==feval(f,x,varargin{:}); fpm(2:end-l) == 2*fpm(2:end-l); Isic==H*sum(fpm) /6; x==linspace(a+H /2,b-H /2, M); fpm==feval(f,x,varargin{:}); Isic == Isic+2*H*sum(fpm)/3;

Remark 4.1 (Interpolatory quadratures) All quadrature formulae introduced thus far are remarkable instances of a more general quadrature formula of the form:

J Iappr(f) = 'Lajf(Yj) j==1

(4.25)

quadl quad8

The real numbers Qj are the quadrature weights, while the points Yj are the quadrature nodes. In general, one requires that (4.25) integrates exactly at least a constant function: this property is ensured if E f 1 aj = (b - a). By properly chosing the weights and the nodes, the order of accuracy of formula (4.25) can be made arbitrarily large. For instance, using the MAT- LAB instruction quadl(fun,a,b) (or the obsolete one quadS), it is possible to compute an integral with a high order Gauss-Lobatto quadrature formula. The function fun can be an inline object. For instance, to integrate f(x) == l/x over [1, 2], we must first define the following function fun

```
fun==inline('l./x', 'x');
```

then call quadl (fun, 1,2). Note that in the definition of the function f we have used an element by element operation (indeed MATLAB will evaluate this expression component by component on the vector of quadrature nodes). The specification of the number of subintervals is not requested as it is automatically computed in order to ensure that the quadrature error is below the default tolerance of 10- 3 . A different tolerance can be provided by the user through the extended command quadl(fun,a,b,tol). In Section 4.3 we will introduce a method to estimate the quadrature error and, consequently, to change H adaptively.

. !..: ...

· :.... ....i. :."::..

I.'

'.

.i

i0:,

1. A quadrature formula is a formula to approximate the integral of continuous functions;

4.3 Simpson adaptive formula 95

2. it is generally expressed as a linear combination of the values of the function at specific points (called nodes) with coefficients which are called weights; 3. the degree of exactness of a quadrature formula is the highest de- gree of the polynomials which are integrated exactly by the for- mula. The degree of exactness is 1 for midpoint and trapezoidal formulae, 3 for Gauss and Simpson formulae;

4. the order of accuracy of a composite quadrature formula is its order with respect to the size H of the subintervals. The order of accuracy is 2 for composite midpoint and trapezoidal formulae, 4 for composite Gauss and Simpson formulae.

See the Exercises 4.12-4.18.

The integration step-length H of a quadrature composite formula can be chosen in order to ensure that the quadrature error is less than a pre- scribed tolerance E > O. For instance, when using the Simpson composite formula, this goal can be achieved by virtue of (4.22), if b - a H 4 - max If(4)(x)1 < E, 180 16 xE[a,b]

(4.26)

where f(4) denotes the fourth-order derivative of f. Unfortunately, when the absolute value of f(4) is large only in a small part of the integra- tion interval, the maximum H for which (4.26) holds true can be too small. The goal of the adaptive Simpson quadrature formula is to yield an approximation of I (f) within a fixed tolerance E by a non uniform distribution of the integration step-sizes in the interval [a, b]. In such a way we retain the same accuracy of the composite Simpson rule, but with a lower number of quadrature nodes and, consequently, a reduced number of evaluations of f. To this end, we must find an error estimator and an automatic proce- dure to modify the integration step-length H, according to the achieve- ment of the prescribed tolerance. We start by analyzing this procedure, which is independent of the specific quadrature formula that one wants to apply. In the first step of the adaptive procedure, we compute an approxima- tion 1s(f) of 1(f) = J: f(x) dx. We set H = b-a and we try to estimate

.

. '. :

:::-.'. : .;," . : . . : . .'·'·. ·'·'·. "'·'·'·'·'·'·'·'·.. . ."0/"""",..

the quadrature error. If the error is less than the prescribed tolerance, the adaptive procedure is stopped; otherwise the step-size H is halved until the integral J:+ H f(x) dx is computed with the prescribed accu- racy. When the test is passed, we consider the interval (a + H, b) and we repeat the previous procedure, choosing as the first step-size the length b - (a + H) of that interval. We use the following notations:

1. A: the active integration interval, i.e., the interval where the integral is being computed; 2. S: the integration interval already examined, for which the error is less than the prescribed tolerance;

3. N: the integration interval yet to be examined. At the beginning of the integration process we have N == [a, b], A == N and S == 0, while the situation at the generic step of the algorithm is depicted in Figure 4.5. Let Js(f) indicate the computed approximation of Ja a f(x)dx, with Js(f) = 0 at the beginning of the process; if the algo- rithm successfully terminates, J s (f) yields the desired approximation of 1(f). We also denote by J(a,{3) (f) the approximate integral of f over the active interval [a, /3]. This interval is drawn in gray in Figure 4.5. The generic step of the adaptive integration method is organized as follows: 1. If the estimation of the error ensures that the prescribed tolerance is satisfied, then:

(i) Js(f) is increased by J(a,{3) (f), that is Js(f)

Js(f) + J(a,{3) (f); (ii) we let S

SUA, A == N (corresponding to the path (I) in Figure 4.5) and a

/3 and /3

b;

2. If the estimation of the error fails the prescribed tolerance, then: (j) A is halved, and the new active interval is set to A == [a, a /] with a ' == (a + /3)/2 (corresponding to the path (II) in Figure 4.5); (jj) we let N

N U [a', /3], /3

a ' ; (jjj) a new error estimate is provided. Of course, in order to prevent the algorithm from generating too small step-sizes, it is convenient to monitor the width of A and warn the user, in case of an excessive reduction of the step-length, about the presence of a possible singularity in the integrand function. The problem now is to find a suitable estimator of the error. To this end, it is convenient to restrict our attention to a generic subinterval

r s t

A f.. ,.."""..."........!Y.,,,,,,,,, ",.." ",t

(I)

a I

s

a

A

b I

(II)

a S a A a ' N b A"".....j

Fig. 4.5. Distribution of the integration intervals at the generic step of the adaptive algorithm and updating of the integration grid

[a, jJ] in which we compute I8(f): of course, if on this interval the error is less than E (jJ - a) j (b - a), then the error on the interval [a, b] will be less than the prescribed tolerance E. Since from (4.24) we get

5 ! f(x) dx - 1s(f) = - ((32;8

) f(4)(

) = Es(f;a,{3), a

to ensure the achievement of the tolerance, it will be sufficient to verify that E8(f; a, jJ) < E(jJ - a)j(b - a). In practical computation, this procedure is not feasible since the point

E [a, jJ] is unknown. To estimate the error without using explicitly the value f(4) (

), we employ again the composite Simpson formula to compute J: f(x) dx, but with a step-length (jJ - a)j2. From (4.22) with a == a and b == jJ, we deduce that

5 ! f(x) dx - 1

(f) = - (

;J f(4) (7]),

(4.27)

a

where TJ is a suitable point different from

. Subtracting the last two equations, we get

!:i1 = 1C(f) - I (f) = _ ({3 - a)5 f(4)(

) + ({3 - a)5 f(4) (7]). (4.28) 8 8 2880 46080 Let us now make the assumption that f(4)(x) is approximately a con- stant on the interval [a, jJ]. In this case f(4)(

) rv f(4)(TJ). We can com- pute f(4)(7]) from (4.28) and, putting this value in the equation (4.27), we obtain the following estimation of the error:

(3 ! f(x) dx - 1

(f) r--J

!:i1. 15 a

The step-length ((3-a)/2 (that is the step-length employed to compute I

(f)) will be accepted if I

II/15 < E((3 - a)/[2(b - a)]. The quadrature formula that uses this criterion in the adaptive procedure described pre- viously, is called adaptive Simpson formula. It is implemented in Program 6. Among the input parameters, f is the string in which the function f is defined, a and b are the endpoints of the integration interval, tol is the prescribed tolerance on the error and hmin is the minimum

admissible value of the integration step-length (in order to ensure that the adapt ion procedure always terminates).

function [JSf,nodes]==simpadpt(f,a, b,tol ,hmin) %SIMPADPT Numerically evaluate integral, adaptive Simpson quadrature. % JSF == SIMPADPT(FUN,A,B,TOL,HMIN) tries to approximate the % integral of function FUN from A to B to within an error of TOL using % recursive adaptive Simpson quadrature. The inline function Y == FUN(V) % should accept a vector argument V and return a vector result Y, the % integrand evaluated at each element of X. % % [JSF,NODES] == SIMPADPT(...) returns the distribution of nodes. A==[a,b]; N==[]; S==[]; JSf == 0; ba == b - a; nodes==[]; while - isempty(A), [delta I, ISc] ==caldelta i(A, f); if abs(deltal) <== 15*tol* (A(2)-A(l))jba; JSf == JSf + ISc; S == union(S,A); nodes == [nodes, A(l) (A(l)+A(2))*0.5 A(2)]; S == [S(l), S(end)]; A == N; N == []; elseif A(2)-A(l) < hmin JSf==JSf+ISc; S == union(S,A); S == [S(l), S(end)]; A==N; N==[]; warning('Too small step-length'); else Am == (A(l)+A(2))*0.5; A == [A(l) Am]; N == [Am, b]; end end nodes== un iq ue(nodes) ; return

function [deltal, ISc]==caldeltai(A, f) L==A(2)-A(l);

4.4 What we haven't told you 99

t==[O; 0.25; 0.5; 0.5; 0.75; 1]; x==L *t+A(1); L==L/6; w== [1; 4; 1]; fx==feva I(f,x); IS==L *sum(fx([1 3 6]). *w); ISc==0.5*L *sum(fx. * [w;w]); deltal==IS-ISc; return

Example 4.3 Let us compute the integral 1(1) = J

l e- 1O (x-l)2 dx by the adaptive Simpson formula. Running Program 6 with tol == 10- 4 and hmin == 10- 3 provides 0.28024765884708, instead of the exact value 0.28024956081990. The error is less than the prescribed tolerance tol=10- 5 . To obtain this result it was sufficient to use only 10 nonuniform subintervals. Note that the corresponding composite formula with uniform step-size would have required 22 subintervals to ensure the same accuracy.

The midpoint, trapezoidal and Simpson formulae are particular cases of a larger family of quadrature rules known as N ewton- Cotes formulae. For an introduction, see [QSSOO], Chapter 10. Analogously, the Gauss formula (4.20) is a special case of the important family of Gaussian quadrature formulae. These are optimal in the sense that they maxi- mize the degree of exactness for a given number of quadrature nodes. In MATLAB 6 the function quadl implements one of these formulae. For an introduction to the Gaussian formulae, see [QSSOO], Chapter 10 or [RR85]. Further developments on numerical integration can be found, e.g., in [DR75] and [PdDKUK83]. Numerical integration can also be used to compute integrals on un- bounded intervals. For instance, to approximate Jo oo f(x) dx, a first pos- sibility is to find a point a such that the value of J: f (x)dx can be neglected with respect to that of Jo a f(x)dx. Then we compute by a quadrature formula this latter integral on a bounded interval. A second possibility is to resort to Gaussian quadrature formulae for unbounded intervals (see [QSSOO], Chapter 10). Finally, numerical integration can also be used to compute multidi- mensional integrals. In particular, we mention the MATLAB instruction dblquad (, f ' , xmin, xmax , ymin, ymax) by which it is possible to com- dblquad pute the integral of a function contained in the MATLAB file f . mover the rectangular domain [xmin, xmax] x [ymin, ymax]. Note that the

function f must have at least two input parameters corresponding to the variables x and y with respect to which the integral is computed.

Exercise 4.1 Verify that, if f E 0 3 in a neighborhood 10 of Xo (respectively, In of x n) the error of the formula (4.10) is equal to -

f"' (

0)h2 (respectively, -

f"' (

n)h2), where

o and

n are two suitable points belonging to 10 and In, respectively.

Exercise 4.2 Verify that if f E 0 3 in a neighborhood of x the error of the formula (4.8) is equal to (4.9).

Exercise 4.3 Compute the order of accuracy with respect to h of the follow- ing formulae for the numerical approximation of f' (Xi):

b.

-11f(xi) + 18f(Xi+l) - 9f(Xi+2) + 2f(Xi+3) 6h f(Xi-2) - 6f(Xi-l) + 3f(Xi) + 2f(Xi+l) 6h f(Xi-2) - 8f(Xi) + 8f(Xi+l) - f(Xi+2) 12h

a.

c.

Exercise 4.4 The following values represent the time evolution of the number n(t) of individuals of a given population whose birth rate is constant (b == 2) and mortality rate is d(t) == O.Oln(t):

t (months) n

o 0.5 1 100 147 178

1.5 2 2.5 3 192 197 199 200

Use these data to approximate as accurately as possible the rate of variation of this population. Then compare the obtained results with the exact rate n' (t) == 2n(t) - 0.01n 2 (t).

Exercise 4.5 Find the minimum number M of subintervals to approximate with an absolute error less than 10- 4 the integrals of

the following functions:

1 !I (x) = 1 + (x _ 1f)2 over (0,5), f2(x) = eX cos(x) over (0,1f), 1 13 (x) = vi x (1 _ x) over (0, 1),

using the midpoint composite formula. Verify the results obtained using the Program 3.

Exercise 4.6 Prove (4.13) starting from (4.14).

4.5 Exercises 101

Exercise 4.7 Why does the midpoint formula lose one order of convergence when used in its composite mode?

Exercise 4.8 Verify that, if f is a polynomial of degree less than or equal 1, then Imp(f) == I(f) i.e. the midpoint formula has degree of exactness equal to 1.

Exercise 4.9 For the function fl of Exercise 4.5, compute (numerically) the values of M which ensure that the quadrature error is less than 10- 4 when the integral is approximated by the composite trapezoidal and Gauss quadrature formulae.

Exercise 4.10 Let II and 12 be two values obtained by the composite trape- zoidal formula applied with two different step-lengths, HI and H 2 , for the approximation of 1(1) = J: f(x)dx. Verify that, if f(2) has a mild variation on (a, b), the value

IR == II + (II - I2)/(H

I Hi - 1)

(4.29)

is an approximation of I(f) better than II and 12. This strategy is called the Richardson extrapolation method.

Exercise 4.11 Verify that, among all formulae of the form Iappx(f) == af(x)+ (3f(z) where x, z E [a, b] are two unknown nodes and a and (3 two undeter- mined weights, formula (4.20) features the maximum degree of exactness.

Exercise 4.12 For the first two functions of Exercise 4.5, compute the min- imum number of intervals such that the quadrature error of the composite Simpson quadrature formula is less than 10- 4 .

Exercise 4.13 Compute J; e- x2 / 2 dx by the Simpson formula (4.23) and Gauss formula (4.20), then compare the obtained results.

Exercise 4.14 To compute the integrals h = J 0 1 x k e x - 1 dx for k = 1,2,..., one can use the following recursive formula: Ik == 1 - kI k - 1 , with II == lie. Compute 1 20 by the composite Simpson formula in order to ensure that the quadrature error is less than 10- 3 . Compare the Simpson approximation with the result obtained by the above recursive formula.

Exercise 4.15 Apply the Richardson extrapolation formula (4.29) for the approximation of the integral 1(1) = J 0 2 e- x2 / 2 dx, with HI = 1 and H2 = 0.5 using first Simpson formula (4.23), then the Gauss formula (4.20). Verify that I R is more accurate than II and 1 2 .

Exercise 4.16 Compute by the composite Simpson formula the function j(r) defined in (4.2) for r == kilO m with k == 1, . . . , 10, with p(

) == e

and a == 0.36 W I(mK). Ensure that the quadrature error is less than 10- 10 . (Recall that: m=meters, W ==watts, K==degrees Kelvin.)

Exercise 4.17 By using the composite Simpson and Gauss formulae compute the function E(T), defined in (4.1), for T equal to 213 K, up to at least 10 exact significant digits.

Exercise 4.18 Develop a strategy to compute 1(1) = IIG Ix 2 - 0.251 dx by the composite Simpson formula such that the quadrature error

is less than 10- 2 .

5. Linear systems

In applied sciences, one is quite often led to face a linear system of the form

Ax == b

(5.1)

for the solution of complex problems, where A is a square matrix of dimension n x n whose elements aij are either real or complex, while x and b are column vectors of dimension n with x representing the unknown solution and b is a given vector. Component-wise, (5.1) can be written as

allxl + al2 X 2 + . · · + alnX n == b l ,

a2l x l + a22 x 2 + · · · + a2n X n == b 2 ,

anl Xl + a n 2 X 2 + · · · + annX n == b n ,.

w

present three different problems that give rise to linear systems.

Problem 5.1 (Hydraulics) Let us consider the hydraulic network made of the 10 pipelines in Figure 5.1, which is fed by a reservoir of water at constant pressure pr == 10 bar. In this problem, pressure values refer to the difference between the real pressure and the atmospheric one. For the j-th pipeline, the following relationship holds between the flow-rate Qj (in m 3 /s) and the pressure gap

Pj at pipe-ends:

Qj == kL

pj,

(5.2)

where k is the hydraulic resistance (in m 2 / (bar s)) and L is the length (in m) of the pipeline. We assume that water flows from the outlets (indicated by a black dot) at atmospheric pressure, which is set to 0 bar for coherence with the previous convention.

A typical problem consists of determining the pressure values at every inter- nal node 1, 2, 3, 4. With this aim, for each j == 1,2,3,4 we can supplement the relationship (5.2) with the statement that the algebraic sum of the flow-rates of the pipelines which meet at j-th node must be null (a negative value would indicate the presence of a seepage). Denoting by p == (Pl,P2,P3,P4)T the pressure vector at the internal nodes, we get a 4 x 4 system of the form Ap == b. Let us summarize in the following table the relevant characteristics of the different pipelines.

pipeline k L pipeline k L pipeline k L 1 0.01 20 2 0.005 10 3 0.005 14 4 0.005 10 5 0.005 10 6 0.002 8 7 0.002 8 8 0.002 8 9 0.005 10 10 0.002 8

Correspondingly, A and b take the following values (only the first 4 significant digits are provided):

A==

-0.370 0.050 0.050 0.070

0.050 -0.116 o 0.050

0.050 o -0.116 0.050

0.070 0.050 0.050 -0.202

b==

-2 o o o

The solution of this system is postponed to Example 5.4.

.

QI .

p==o

p==o

p==o

Q 6

Fig. 5.1. The pipeline network of Problem 5.1

Problem 5.2 (Spectrometry) Let us consider a gas mixture of n non reac- tive unknown components. By the mass spectrometer the compound is bom- barded by low-energy electrons: the resulting mixture of ions is analyzed by a galvanometer which shows peaks corresponding to specific ratios mass/charge. We consider only the n most relevant peaks. One may conjecture that the height hi of the i-th peak is a linear combination of {Pj, j == 1,..., n}, Pj

being the partial pressure of the j-th component (that is the pressure exerted by a single gas when it is part of a mixture), yielding

n LSijPj = hi, i = 1, . . . , n, j==1

where the Sij are the so-called sensitivity coefficients. The determination of the partial pressures demands therefore the solution of a linear system. .

Problem 5.3 (Economy: input-output analysis) We want to determine the situation of equilibrium between demand and offer of certain goods. In particular, let us consider n factories that produce n different products. They must face the internal demand of goods necessary to the factories for their own production, as well as the external demand from the consumers. Let Xi, i == 1, . . . , n, denote

the number of units of the i-th product of the i-th factory and let b i , i == 1, . . . , n, denote the number of units of the i-th product absorbed by the market. Finally, let Cij be the fraction of Xi necessary to the j-th factory for the production of the j-th product (see Figure 5.2). According to the Leontief model, we assume that the transformation functions that relate the various problems are linear. Then the equilibrium is reached when the vector x of the total production equals the total demand, that is, x == Cx + b, where C == (Cij) and b == (b i). Thus, in this model the total production satisfies the linear system

Ax == b, where A == I-C.

(5.3)

For its solution, see Exercise 5.17.

.

Cll I

1 - 0 . b l C12 I

C22 00 2 - mJ . b 2 t C3l 3 d -I .]] b 3 t C33 Fig. 5.2. The interaction scheme of 3 factories with the market

The solution of the system (5.1) exists iff A is non singular. In principle, the solution might be computed by the so-called Cramer rule:

det(A i) Xi = det(A) ,

i == 1, . . . , n,

where Ai is the matrix obtained from A by replacing the i-th column by band det(A) denotes the determinant of A. If the n+ 1 determinants are computed by the Laplace expansion (see Exercise 5.1), a total number of 2(n + 1)1 ops is required. As usual, by operation we mean a sum, a subtraction, a product or a division. For instance, a computer capable of carrying out 10 9 flops (i.e. 1 giga ops per second), would require about 12 hours to solve a system of dimension n == 15,3240 years if n == 20 and 10 143 years if n == 100. The computational cost can be drastically reduced to the order of about n 3 . 8 ops if the n + 1 determinants are computed by the algorithm quoted in Example 1.3. Yet, this cost is still too high for the large values of n which often arise in practical applications. We need therefore to look for alternative approaches. Let us note that in general a system cannot be solved by less than n 2 ops. Indeed, if the equations are fully coupled, we should expect that everyone of the n 2 matrix coefficients would be involved in an operation at least once.

Let A be a square matrix of order n. Assume that there exist two suitable matrices L and V, lower triangular and upper triangular, respectively, such that

A == LV.

(5.4)

We call (5.4) an LV-factorization (or decomposition) of A. Should A be non-singular, so are both L and V, and thus their diagonal elements are non-null (as observed in Section 1.3). In such a case, solving Ax == b leads to the solution of two triangular systems

Ly == b, Vx == y.

(5.5)

Both systems are easy to solve. Indeed, L being lower triangular, the first row of the system Ly == b takes the form:

lllYl == b l ,

which provides the value of Yl since III -=I=- o. By substituting this value of Yl in the subsequent n - 1 equations we obtain a new system whose unknowns are Y2, . . . , Yn, on which we can proceed in a similar manner.

Proceeding forward, equation by equation, we can compute all unknowns with the following forward substitutions algorithm:

1 Yl == - z bl, 11

1 Yi == - Zii

i-I b i - LlijYj j==1

(5.6)

i == 2, . . . , n

Let us quantify the number of operations required from (5.6). Since i-I sums, i-I products and 1 division are needed to compute the unknown Yi, the total number of operations required is

n

n

n

L 1 + 2L(i -1) = 2Li - n = n 2 . i==1 i==1 i==1

The system U x == y can be solved by proceeding in a similar manner. This time, the first unknown to be computed is X n then, by proceeding backward, we can compute the remaining unknowns Xi, for i == n - 1 to i == 1:

1 X n == Yn, U nn

1 Xi == - Uii

n Yi - L UijXj j==i+l

(5.7)

i == n - 1, . . . , 1

This is called backward substitutions algorithm and requires n 2 ops too. At this stage we need an algorithm that allows an effective computation of the factors Land U of the matrix A. We illustrate a general procedure starting from a couple of examples.

Example 5.1 Let us write the relation (5.4) for a generic matrix A E IR 2X2

[

l

2] [U

l

] [:

:

].

The 6 unknown elements of L and V must satisfy the following (non-linear) equations:

(el) III Ull == all, (e3) l2lUll ==a2l,

(e2) lllU12 == a12, (e4) l2lU12 + l22U22 == a22.

(5.8)

System (5.8) is underdetermined as it features less equations than unknowns. We can complete it by assigning arbitrarily the diagonal elements of L, for instance setting III == 1 and l22 == 1. Now system (5.8) can be solved by proceeding as follows: we determine the elements U11 and U12 of the first row of U using (e1) and (e2). If U11 is nonnull then from (e3) we deduce l21 (that is the first column of L, since III is already available). Now we can obtain from (e4) the only non zero element U22 of the second row of U.

Example 5.2 Let us repeat the same computations in the case of a 3 x 3 matrix. For the 12 unknown coefficients of Land U we have the following 9 equations:

(e1) lllU11 ==a11, (e4) l21 U11 == a21, (e7) l31U11 == a31,

(e2) lll U12 == a12, (es) l21 U12 + l22U22 == a22, (es) l31 U12 + l32U22 == a32,

(e3) lll U13 == a13, (e6) l21 U13 + l22U23 == a23, (eg) l31 U13 + l32U23 + l33U33 == a33.

Let us complete this system by setting lii == 1 for i == 1,2,3. Now, the coefficients of the first row of U can be obtained by using (e1), (e2) and (e3). Next, using (e4) and (e7), we can determine the coefficients l21 and l31 of the first column of L. Using (es) and (e6) we can now compute the coefficients U22 and U23 of the second row of U. Then, using (e8), we obtain the coefficient l32 of the second column of L. Finally, the last row of U (which consists of the only element U33) can be determined by solving (eg). On a matrix of arbitrary dimension n we can proceed as follows: 1. The elements of L and V satisfy the system of non-linear equations

mine i,j) L lirUrj = aij, i, j = 1, . . · , n; r==l

(5.9)

2. System (5.9) is underdetermined; indeed there are n 2 equations and n 2 +n unknowns, thus the factorization LV cannot be unique; 3. By forcing the n diagonal elements of L to be equal to 1, (5.9) turns into a determined system which can be solved by the following

Gauss algorithm: set A(1) = A i.e. ag) = aij for i,j = 1,... ,n; for k == 1, . . . , n - 1 do

for i == k + 1, . . . , n . (k) a ik lik = (k) ' a kk for j == k + 1, . . . , n (k+l) _ (k) (k) a. . - a.. - Z'; k a k . 'lJ 'lJ fI J

(5.10)

5.1 The LV factorization method 109

The elements ak

must all be different from zero and are called pivot elements. For every k = 1,..., n - 1 the matrix A (k+l) = (a

7+1)) has n - k rows and columns.

At the end of this procedure the elements of the upper triangular · U · b - (i) £ . - 1 d . - . matrIx are gIven y Uij - a ij or 2 - , . . . , n an J - 2,..., n, whereas those of L are given by the coefficients lij generated by this algorithm. In (5.10) there is no computation of the diagonal elements of L, as we already know that their value is equal to 1. This factorization is called the Gauss factorization; the determination of the elements of the factors Land U requires about 2n 3 /3 ops (see Exercise 5.4).

Example 5.3 Consider the following Vandermonde matrix

A ( ) · h (n-j).. 1 == aij WIt aij == Xi ,

, J == ,..., n,

(5.11)

where the Xi are n distinct abscissae. It can be constructed using the MAT- LAB command vander. In Figure 5.3 we report the number of floating poin

vander operations required in order to compute the Gauss factorization of A. Sev- eral values of n are considered in abscissae, and the corresponding number of ops are indicated with circles. They have been obtained using the following commands:

> > ops == [ ]; for n == [10: 10: 100] A == vander(linspace(1,2,n)); flops(O), A == lu_gauss(A); ops== [ops, flops]; end

The curve reported in the picture is a polynomial in n of the third order representing the least square approximation of the above data, and is generated by the following commands:

> > c3==polyfit([10: 10: 100] ,ops,3) 0.6667 -0.0000 0.3333 -0.0000

,The coefficient of the monomial n 3 is exactly 2/3.

It is not necessary to store the matrices {A (k)}; actually we can overlap the (n - k) x (n - k) elements of A (k+ 1) on the corresponding last (n - k) x (n-k) elements of the original matrix A. Moreover, since at the k-th step, the sub diagonal elements of the k-th column don't have any effect on the final U, they can be .replaced by the entries of the k-th column of L, as done in Program 7. Then, at the k-th step of the process the.

5 7 x10

1 ..... . ,-..,.....,.,

, , , . .

. , . , . , , , . , .. "

6 ",,",,""

5 ",,",,'

4 """""" . "-,.,.,.,.,..,,, ",.,.,."-..,.,,,.-

3 """",

2 """",," """" .." ... "_...",,,

00

20

40

60

80

100

Fig. 5.3. The number of ops as a function of n, necessary to generate the Gauss factorization LV of the Vandermonde matrix. This function is a cubic polynomial obtained by approximating in the least square sense the values (represented by circles) corresponding to n == 10, 20, . . . , 100

elements stored at location of the original entries of A are (1) (1) (1) all a 12 a 1n (2) (2) l21 a 22 a 2n

lk1

lk k-1 ,

(k) a kk

(k) a kn

ln1

ln k-l ,

(k) an,k

(k) ann

where the boxed submatrix is A (k). The Gauss factorization is the basis of several MATLAB commands: lu [L, U] =lu (A) whose mode of use will be discussed in Section 5.2; lnv in v that allows the computation of the inverse of a matrix;

\

\

by which it is possible to solve a linear system with matrix A and right hand side b by simply writing A \ b.

Remark 5.1 (Computing a determinant) By means of the LV factoriza- tion one can compute the determinant of A with a computational. cost of O(n 3) operations, noting that (see Sect.I.3)

n det(A) = det(L) det(U) = II Ukk. k==I

As a matter of fact, this procedure is also at the basis of the MATLAB com- mand det. det

In Program 7 we implement the algorithm (5.10). The factor L is stored in the (strictly) lower triangular part of A and U in the upper triangular part of A (for the sake of storage saving). After the program execution, the two factors can be recovered by simply writing: L = eye(n) + tril(A,-i) and U = triu(A), where n is the size of A.

function A==lu_gauss(A) %LU_GAUSS LU factorization without pivoting. % A == LU_GAUSS(A) stores an upper triangular matrix in the upper triangular % part of A and a lower triangular matrix in the strictly lower part of A % (the diagonal elements of L are 1). [n, m] ==size(A); if n - m; error('A is not a square matrix'); else for k == 1:n-1 for i == k+1:n A(i,k) == A(i,k)/A(k,k); if A(k,k) ==== 0, error('Null diagonal element'); end for j == k+1:n A (i ,j) == A (i ,j) - A (i , k) * A (k ,j) ; end end end end

Example 5.4 Let us compute the solution of the system encountered in the Problem 5.1 by using the L U factorization, then applying the backward and forward substitution algorithms. We need to compute the matrix A and the right hand side b and execute the following instructions:

> > A==lu_gauss(A); » y(l)==b(l); for i == 2:4; y == [y; b(i)-A(i,1:i-1)*y(1:i-1)]; end » x(4)==y(4)/A(4,4); » for i == 3:-1:1; x(i)== (y(i)-A(i,i+1:4)*x(i+l:4))/A(i,i); end The result is p == (8.1172,5.9893,5.9893, 5.7779)T.

Example 5.5 Suppose that we solve Ax == b with 1 1-c 3 5-c A== 2 2 2 b== 6 c E ffi., (5.12) 3 6 4 13

whose solution is x == (1,1, I)T (independently of the value of E). Let us set E == 1. The Gauss factorization of A obtained by the Program 7 yields

100 L== 2 1 0 331

1 0 3 U == 0 2 -4 o 0 7

If we set E == 0, despite the fact that A is non singular, the Gauss factoriza- tion cannot be carried out since the algorithm (5.10) would involve divisions by o.

The previous example shows that, unfortunately, the Gauss factoriza- tion A==L U does not necessarily exist for every non singular matrix A. In this respect, the following result can be proven:

Proposition 5.1 For a given matrix A E JRnxn, its Gauss factoriza- tion exists if and only if the principal submatrices Ai of A of order i == 1,..., n - 1 (that is those obtained by restricting A to its first i rows and columns) are non singular. This factorization is unique if A is non singular.

Going back to Example 5.5, we can notice that when E == 0 the second principal submatrix A 2 of the matrix A is singular. We can identify special classes of matrices for which the hypotheses of Proposition 5.1 are fulfilled. In particular, we mention: 1. Symmetric and positive definite matrices. A matrix A E JRnxn is positive definite if

\Ix E JRn with x i- 0,

x T Ax > 0;

2. Diagonally dominant matrices. A matrix is diagonally dominant by row if

n laiil > Llaijl, i=l,...,n, j=l j¥-i

by column if

n l ai i l > L l a j ii, i = 1, · · · , n. j=1 j¥-i

A special case occurs when in the previous inequalities we can replace > by >. Then the matrix A is called strictly diagonally dominant (by row or by column, respectively).

If A is symmetric and positive definite, it is moreover possible to construct a special factorization:

A == HH T ,

(5.13)

where H is a lower triangular matrix with positive diagonal elements. This is the so-called Cholesky factorization and requires about n 3 /3 op- erations (half of those required by the Gauss LV factorization). Further, let us note that, due to the symmetry, only the lower part of A is stored, and H can be stored in the same area. The elements of H can be computed by the following algorithm: for i == 1, . . . , n,

(j-l) h ij = hI.. aij - Lhikhjk , JJ k===1

j == 1,...,i-1,

i-I h ii = aii - Lh;k O k===1

Cholesky factorization is available in MATLAB by setting H=chol (A). chol

See the Exercises 5.1-5.5.

. ,:. 'Wi:.

.

We are going to introduce a special technique that allows us to achieve the LV factorization for every non singular matrix, even if the hypotheses of Proposition 5.1 are not fulfilled. Let us go back to the case described in Example 5.5 and take € == O. Setting A (1) == A after having carried out the first step (k == 1) of the procedure, the new entries of A are

1 1 3 2 0 -4 3 3 -5

(5.14)

Since the pivot a22 is equal to zero, this procedure cannot be continued further. On the other hand, should we interchange the second and third rows beforehand, we would obtain the matrix

1 1 3 3 3 -5 2 0 -4

and thus the factorization could be accomplished without involving a division by o. We can state that permutation in a suitable manner of the rows of the original matrix A would make the entire factorization procedure feasible even if the hypotheses of Proposition 5.1 are not verified. Unfortunately, we cannot know a priori which rows should be permuted. However, this decision can be made at every step k at which a null diagonal element ak

is generated. Let us return to the matrix A(2) in (5.14): since ag) is null, let us interchange the third and second row of A (2) and check whether the new a

;) that is generated is still null. By executing the second step of the factorization procedure we find the same matrix that we would have generated by an a priori permutation of the same two rows of A. We can therefore perform a row permutation as soon as this becomes necessary, without carrying out any a priori transformation on A. Since a row permutation entails changiIlg the pivot element, this technique is given the name of pivoting by row. The factorization that is generated in this way returns the original matrix up to a row permutation. Precisely we obtain

PA == LU.

P is a suitable permutation matrix. At the beginning it is set equal to the identity matrix, then if in the course of the procedure the rows r

and s of A are permuted, the same permutation must be performed on the homologous rows of P. Correspondingly, we should now solve the following triangular systems

Ly == Pb, Ux == y.

(5.15)

From the second equation of (5.10) we see that not only null pivot ele- ments ak

are troublesome, but also those which are very small. Indeed, should ak

be near zero, possible roundoff errors affecting the coefficients ak

) will be severely amplified.

Example 5.6 Consider the nonsingular matrix

1 1 + 0.5 . 10- 15 3 A == 2 2 20 3 6 4

During the factorization procedure by Program 7 no null pivot elements are obtained. Yet, the factors Land U turn out to be quite inaccurate, as one can

realize by computing the residual matrix A - LU (which should be the null matrix if all operations were carried out in exact arithmetic):

000 A - LU == 0 0 0 006

It is therefore recommended to carry out the pivoting at every step of the factorization procedure, by searching among all virtual pivot elements a

) with i = k,..., n, the one with maximum modulus. The algorithm (5.10) with pivoting by row carried out at every step takes the following form: for k == 1, . . . , n, do

for i == k + 1, . . . , n find f such that la

%) I = max la

I, r

k,...,n exchange row k with row f,

(k) aO k lik = (k) ' a kk for j == k + 1, . . . , n (k+ 1) (k) (k) aij == aij - likakj

(5.16)

The MATLAB program lu that we have mentioned previously computes the Gauss factorization with pivoting by row. Its complete syntax is indeed [L, U, P] =lu (A), P being the permutation matrix. When called in the shorthand mode [L, U] =lu (A) , the matrix L is equal to P*M, where M is lower triangular and P is the permutation matrix generated by the pivoting by row. The program lu activates automatically the pivoting by row when a null (or very small) pivot element is computed.

See the Exercises 5.6-5.8.

.

. .

We have already noticed in Example 5.6 that, due to roundoff errors, the product LV does not reproduce A exactly. Even though the pivoting strategy damps these errors, yet the result could sometimes be rather unsatisfactory.

Example 5. 7 Consider the linear system Anxn = b n where An E JRn X n is

the so-called Hilbert matrix whose elements are .

aij

1/ (i + j - 1),

i, j

1, . . . , n,

while b n is chosen in such a way that the exact solution is X n

(1, 1, . . . , 1) T. The matrix An is clearly symmetric and one can prove that it is also positive definite. For different values of n we use the MATLAB function lu to get the Gauss factorization of An with pivoting by row. Then we solve the associated linear systems (5.15) and denote by x n the computed solution. In Figure 5.4 we report (in logarithmic scale) the relative errors

En

IIXn - X n II/llx n II,

(5.17)

having denoted by II . II the Euclidean norm introduced in the Section 1.3.1. We have En > 10 if n > 13 (that is a relative error on the solution higher than 1000%!), whereas Rn

Ln Un - PnAn is the null matrix (up to machine accuracy) for whatever value of n.

10 5

10°

10- 5

10- 10

--.,-

--

----------_.

1 0- 20 o

20

40

60

80

100

Fig. 5.4. Behavior versus n of En (solid line) and ofmaXi,j==l,...,n Irijl (dashed line) in logarithmic scale, for the Hilbert system of Example 5.7. The (rij) are the coefficients of the matrix R

On the ground of the previous remark, we could speculate by saying that, when a linear system Ax == b is solved numerically, one is indeed looking for the exact solution x of a perturbed system

(A + l5A)x == b + t5b,

(5.18)

where l5A and t5b are respectively a matrix and a vector which depend on the specific numerical method which is being used. We start by con- sidering the case where l5A == 0 and t5b -I 0 which is simpler than the most general case. Moreover, for simplicity we will also assume that A is symmetric and positive definite. By comparing (5.1) and (5.18) we find x - x == -A -lt5b, and thus

II x - xii == IIA -lt5bll.

(5.19)

In order to find an upper bound for the right hand side of (5.19), we proceed as follows. Since A is symmetric and positive definite, the set of its eigenvectors {Vi} f 1 furnishes an orthonormal basis of JRn (see [QSSOO], chapter 5). This means that

AVi == AiVi,

i == 1,..., n,

T _

vi Vj - Vij,

i,j == 1,...,n,

where Ai is the eigenvalue of A associated with Vi and 6ij is the Kronecker symbol. Consequently, a generic vector w E JRn can be written as

n W = LWiVi, i==1

for a suitable (and unique) set of coefficients Wi E JR. We have

IIAwl1 2 == (Aw)T(Aw) == [Wl(Avl)T + ... + Wn(Avn)T] [WI Av l + ... + wnAv n] == (AI wI vi + · · · + AnWn V

)(AI WI VI + · · · + AnWn Vtn) n = LATW;' i==1

Denote by Amax the largest eigenvalue of A. Since Ilwl1 2 == I:

1 w;, we conclude that

!lAw11 < AmaxllWl1 Vw E JRn.

(5.20)

In a similar manner, we obtain

1 IIA -lwll < Amin Ilwll,

upon recalling that the eigenvalues of A -1 are the reciprocals of those of A. This inequality enables us to draw from (5.19) that

llx - xii 1 III5bll < . II xii - Amin II xii

(5.21)

Using (5.20) once more and recalling that Ax == b, we finally obtain

llx - xii Amax III5bll < II xii - Amin Ilbll

(5.22)

We can conclude that the relative error in the solution depends on the relative error in the data through the following constant (> 1)

K(A) = Amax Amin

(5.23)

which is called spectral condition number of the matrix A. K(A) can be cond computed in MATLAB using the command cond (A). Other definitions for the condition number are available for non symmetric matrices, see [QSSOO], Chapter 3.

Remark 5.2 The MATLAB command cond(A) allows the computation of the condition number of any type of matrix A, even those which are not sym- condest metric and positive definite. A special command condest (A) is available to rcond compute an approximation of the condition number of A, and one rcond (A) for its reciprocal, with a substantial saving of floating point operations. If the matrix A is ill-conditioned (i.e. K(A) » 1), the computation of its condition number can be very inaccurate. Consider for instance the tridiagonal matrices An == tridiag(-1,2, -1) for different values of n. An is symmetric and pos- itive definite, its eigenvalues are Aj == 2 - 2 cos(jO), for j == 1,..., n, with o == 1r/(n+ 1), hence K(An) can be computed exactly. In Figure 5.5 we report the value of the error EK(n) == IK(An) - cond(An)l/ K(A n). Note that EK(n) increases when n increases.

-10 10 :::::::::::::::::;:::::::::::::::::::::: .::::::::::::::;::::;:::::

....,...."...,.....,.....,..",. " ..,..,.....,..
1-.'....."".

..."...,.....-,.,."....,.... ,",.....,..... . .

. . "....", """" . .

. . . , . , , ... , ...

. . . . . , ... , . , . . , ...

,...,.....-e.....,.....,....., . . .

,....."..... r..... ,",.....,..... . .

..... ,.,....."""""""", ...

-11 : : 10 :::::::::::::::::;:::::::::::::::::::';:::::::::::::: ". """.. I "'...0,.'
'.00.."".,.,. ,.."".. I """". I

.”” I....'..... .,.”., I ,”,' ””””,.....', . ,,'.. 0....”... 0...'.0..... , I.'..”.. 0..... 0.....'....,,,,,.

. . '0”'.'....,,'....., 0'.....”” I ”” - . , , , , , , , , , . . , , , , , ... 0 , , 0 , , , ... , , , . '_' . , , , , , . , , , , , , , , , , , , , ... , , , , , , , . . . , . , , ... , , , . , . . . , . . , , , , , , , . . , .. , , , - - . - ”

. . 10- 12 ::::::::::::::';.;:::::::::::::::::

::::::::::::::::: :

: : : : : : : : : : : : : : : : : : :

: : : : : : : : : : : : : : : : : : :

: : : "0" ...", '0"0" 0.. 0'.....,.. .”0..””,.”””””, .”,...,.....”.....”.....,.”””””,..

. . . - - ”,. ..,...”.”,....,.,..,..,'.....,.””...”,.”.”.,.,..”.. 0'.' 0.....””..0.””.,.”” - . . - .

, , ,. ”””” ,.,.,.....,.....,.”” ...,...,.,.”” ,...,.”” , ' . ' , , , ... , . . , . , . . '.. . 0 . 0 0 , . . 0 0 . . , . , , ... , ... ”” .”.””.,..,.,...,.....,....,.....0.”””””,.,'. ,.””, 0... 0'...”,. .”.....,....”..,..”,..

.. . -13 . . . . . 10 ::: : : : : : : : : : : : : : : i: :::: : : : : : : : : : : : : : : : :> : : : : : : : : : : : : : : : : : : :

0""""""..,'..', 0.'."....". 0.....'.....,...".,... . .

. , , ... , ... , . 0 . . . . , , , 4 . . , . .

-14 . . 10 :::::::::::::::: i:::::::::::::::: i::::::::::::::

..,.,.,..... .",....."..".."..,,,,.. . . . . .

...,,'..... 0.....' 0 0.'..... O' .."..,....."".....,...."""., 0 . .
""".... ",.....,..... 0""""""""

,."""..,.,. 0.. r'".'"... 0.'.....,...."., .."... ".""".".... 0....0..'".. 0.'.'.....,.".""",.,.

....,.,.....,.,.....,... ."",." ,..."",.,. . .

"",.....,..... ,-...,...."..".".""". . .

"""".0.'"....,,.. 0...'.....,.....,.,.".."",...., . .

,.."," 0.....,...,.....,...,.....,.,.,."..",.., """",.....,.....",...,..",.,.,.",.. ". -" -15
:: .. 10 ::::::::::::::::;:::::::::::::::::: ::::::::::::::::::::::::::::::::;.....:

:::::::::::::::::::;::: ......"."..............,."..... 0..... 0'.'.....,. 0....00....
0" 0 0 0....." 0""""

.....,...,.,....".... 0 0.0.000..... .-....."". 0"""" .-0.....0..0. 0..... .-..
0..'....0.....',."... - - " " .

."....", 0...'..".. 0..... 0.....,....,.. 0"" 0.'....., '....,.....0.....
0..0.....,.....,..,., - - " " .

10- 16 o

...,....."....,.,.....,.....,..... 0.....,.....,..... .-0'..... 0'.....".... 0.....,.,.."...
- . " " . - " - . .

500

1000

1500

2000

2500

Fig. 5.5. Behavior of EK(n) as a function of n (in logarithmic scale)

A more involved proof would lead to the following more general result in the case where JA is an arbitrary symmetric and positive definite

matrix "small enough" to satisfy Amax(8A) < Amin(A):

llx - xii K(A) (Amax(8A) 118b ll) IIxll < 1 - Amax(8A)/Amin Amax + IIbll ·

If K(A) is "small", that is of the order of the unity, A is said to be well conditioned. In that case, small errors in the data will lead to errors of the same order of magnitude in the solution. This could not occur in the case of ill conditioned matrices.

Example 5.8 For the Hilbert matrix introduced in Example 5.7, K(An) is a rapidly increasing function of n. One has K(A 4) > 15000, while if n > 13 the condition number is so high that MATLAB warns that the matrix is "close to singular". Actually, K(An) grows at an exponential rate: K(An)

e 3 . 5n (see, [Hig96]). This provides an indirect explanation of the bad results obtained in Example 5.7.

Inequality (5.22) can be reformulated by the help of the residual r:

r == b - Ax.

(5.24)

Should x be the exact solution, the residual would be the null vector. Thus, in general, r can be regarded as an estimator of the error x-x. The extent to which the residual is a good error estimator depends on the size of the condition number of A. Indeed, observing that 8b == A(x - x) == Ax - b == -r, we deduce from (5.22) that

llx - xii < K(A) llrll llxil - llbll

(5.25)

Thus if K (A) is "small" , we can be sure that the error is small provided that the residual is small, whereas this might not be true when K(A) is "large".

Example 5.9 The residuals associated with the computed solution of the linear systems of Example 5.7 are very small (their norms vary between 10- 16 and 10- 11); however the computed solutions differ remarkably from the exact solution.

See the Exercises 5.9-5.10.

.,. . ... - .-.. .... :.:-:.:.;. .:.:.:.:.:.:-:..... .. .. m ... .- ....

In many applications (see for instance Chapter 8), we have to solve a system whose matrix has the form

al Cl 0 A== e2 a2 Cn-l 0 en an

This matrix is called tridiagonal since the only elements that can be non-null belong to the main diagonal and to the first super and sub diagonals. Assume that the matrix coefficients are real numbers. Then if the Gauss factorization L U of A exists, the factors Land U must be bidiag- onals (lower and upper, respectively), more precisely:

1 0 Q1 Cl 0 (32 1 Q2 L== U== Cn-l 0 (3n 1 0 Qn

The unknown coefficients Qi and (3i can be determined by requiring that the equality L U == A holds. This yields the following recursive relations for the computation of the Land U factors:

Ql==al,

(3i==

ei

Q i == ai - (3i Ci -1, i == 2, · · · , n.

(5.26)

Qi-l

Using (5.26), we can easily solve the two bidiagonal systems Ly == band Ux == y, to obtain the following formulae:

(Ly == b) Yl == b 1 , Yi == b i - (3iYi-l, i == 2,..., n,

(5.27)

(Ux = Y) X n = Yn , Xi = (Yi - CiXi+l) /ai, i = n - 1,...,1. (5.28) Qn

This is known as the Thomas algorithm and allows the solution of the original system with a computational cost of the order of n operations. spdiags The MATLAB command spdiags allows the construction of a tridiag- onal matrix. For instance, the commands

» b==ones(lO,l); a==2*b; c==3*b; » T ==spdiags([b a c],-l:l,lO,lO); compute the tridiagonal matrix T E :[RIO x 10 with elements equal to 2 on the main diagonal, 1 on the first subdiagonal and 3 on the first super- diagonal. Note that T is stored in a sparse mode, according to which the only elements stored are those different than o. When a system is solved by invoking the command \, MATLAB is able to recognize the type of matrix (in particular, whether it has been generated in a sparse mode) and select the most appropriate solution algorithm. In particular, when A is a tridiagonal matrix generated in sparse mode, the Thomas algorithm is the selected algorithm.

Example 5.10 Let us solve the linear system Tx == b where T is a tridiagonal matrix constructed as previously with a variable size n, and b is chosen such that the exact solution is x == (1, . . . , l)T. With the following instructions we solve this system (by invoking the \

command) for increasing values of nand compute the number of ops divided by n. This ratio for n very large tends to a constant, proving that the number of operations scales linearly with n, as expected. » ratio == [ ]; > > for k == 50:25:200 b==ones(k,l); a==2*b; c==3*b; T ==sp d i a gs ([b a c], -1 : 1 , k, k) ; rhs == T*ones(k,l); flops(O); x==T\rhs; ratio == [ratio, flops/k]; end > > ratio == Columns 1 through 7 24.5000 24.6133 24.7300 24.7520 24.8067 24.7771 24.8150

. . . . :..... . ;,,/

1. The factorization LV of A consists of computing a lower triangular matrix L and an upper triangular matrix V such that A == LV;

2. the factorization LV, provided it exists, is not unique. However, it can be determined unequivocally by furnishing an additional condition such as, e.g., setting the diagonal elements of L equal to 1. This is called Gauss factorization; 3. the Gauss factorization exists if and only if the principal subma- trices of A of order 1, 2, . . . , (n - 1) are non singular (otherwise

at least one pivot element is null); further, it is unique provided det(A) =I 0;

4. if a null pivot element is generated, a new pivot element can be obtained by exchanging in a suitable manner two rows (or columns)

of our system. This is the pivoting strategy;

5. the computation of the Gauss factorization requires about 2n 3 /3 operations, and only an order of n operations in the case of tridi-agonal systems;

6. for symmetric and positive definite matrices we can use the Cholesky factorization A == HH T , where H is a lower triangular matrix, and the computational cost is of the order of n 3 /3 operations;

7. the sensitivity of the result to perturbation of data depends on the condition number of the system matrix; more precisely, the accuracy of the computed solution can be low for ill conditioned matrices.

:..

.

An iterative method for the solution of the linear system (5.1) consists of setting up a sequence of vectors x(k) E JRn that converge to the exact solution x, that is

lim x(k) == x , k -+ ex)

(5.29)

for any given initial vector x(O) E JRn. A possible strategy able to realize this process can be based on the following recursive definition

X(k+l) == Bx(k) + g,

k > 0,

(5.30)

where B is a suitable matrix (depending on A) and g is a suitable vector (depending on A and b), which must satisfy the relation

x == Bx + g.

(5.31)

Since x == A -lb this yields g == (I - B)A -lb. Let e(k) == x - x(k) define the error at the k-th step. By. subtracting (5.30) from (5.31),

we obtain

e(k+l) == Be(k) .

For this reason B is called the iteration matrix associated with (5.30). If B is symmetric and positive definite, by (5.20) we have

lle(k+l) ll = llBe(k) ll < p(B) lle(k) ll,

Vk > O.

We have denoted by p(B) the spectral radius of B, that is, the maximum modulus of eigenvalues of B. By iterating the same inequality backward, we obtain

lle(k) ll < [p(B)]k lle(O) ll, k > O.

(5.32)

Thus e(k)

0 as k

00 for every possible e(O) (and henceforth x(O)) provided that p(B) < 1. Actually, this property is also necessary for convergence. Should, by any chance, an approximate value of p(B) be available, (5.32) would allow us to deduce the minimum number of iterations k min that are needed to damp the initial error by a factor E. Indeed, k min would be the lowest positive integer for which [p(B)]k rnin < E. In conclusion, the following result holds:

Proposition 5.2 For an iterative method of the form (5.30) whose iteration matrix satisfies (5.31), convergence for any x(O) holds iff p(B) < 1. Moreover, the smaller p(B), the fewer the number of iterations necessary to reduce the initial error by a given factor.

A general technique to devise an iterative method is based on a splitting of the matrix A, A == P - (P - A), being P a suitable nonsingular matrix (called the preconditioner of A). Then x == p-l(p - A)x + P-1b, which has the form (5.31) provided that we set B == p-l (P - A) == I - p-l A and g == p-l b. Correspondingly, we can define the following iterative method:

p(X(k+l) _ x(k)) = r(k) ,

k > 0,

where r(k) == b - Ax(k) denotes the residual vector at the k-th iteration. A generalization of this iterative method is the following

p(X(k+l) _ x(k)) = akr(k) ,

k>O

(5.33)

where Qk f=- 0 is a parameter that may change at every iteration k. The method (5.33) requires at each step the solution of the linear system

pZ(k) == r(k) ,

(5.34)

then the new iterate is defined by x(k+l) == x(k) + QkZ(k). For that reason the matrix P ought to be chosen in such a way that the computational cost for the solution of (5.34) be quite low (e.g., every P either diagonal or triangular or tridiagonal will serve the purpose).

Let us now consider some special instance of iterative methods which take the form (5.33).

The Jacobi method If the diagonal entries of A are nonzero, we can set P == D, where D is the diagonal matrix of the diagollal entries of A. The Jacobi method corresponds to this choice with the assumption Qk == 1 for all k. Then from (5.33) we obtain

DX(k+l) = b - (A - D)x(k),

k > 0,

or, componentwise,

(k+l) 1 x. == -

aii

n b i - L aijxJ k) j==l,j#i

, i == 1,..., n

(5.35)

where k > 0 and x(O) = (x

O) , x

O) , . . . , x

O))T is the initial vector. The iteration matrix is therefore

o -aI2/ all -a21/ a22 0

-al n / all

-a2n/ a22

B == D-I(D - A) ==

(5.36)

-anl/ann -a n 2/ a nn

o

The following result allows the verification of Proposition 5.2 without explicitly computing p(B):

Proposition 5.3 If the matrix A is strictly diagonally dominant by row, then the Jacobi method converges.

As a matter of fact, we can verify that p(B) < 1, where B is given in (5.36). To start with, we note that the diagonal elements of A are nonnull owing to the strict diagonal dominance. Let A be a generic eigenvalue of B and x an associated eigenvector. Then

n Lbijxj = AXi, i = 1,... ,n. j==l

Assume for simplicity that maxk==l,...,n IXkl == 1 (this is not restrictive since an eigenvector is defined up to a multiplicative constant) and let Xi be the component whose modulus is equal to 1. Then

n

n L b..x. < L

J J - aii j==l,j#i j==l,j#i

n

IAI = Lbijxj = j==I

aij

having noticed that B has only null diagonal elements. Therefore IAI < 1 thanks to the assumption made on A. The Jacobi method is implemented in the Program 8 setting in the input parameter P=' J ' . Input parameters are: the system matrix A, the right hand side b, the initial vector xO and the maximum number of iterations allotted, nmax. The iterative procedure is terminated as soon as the ratio between the Euclidean norm of the current residual and that of the initial residual is less than a prescribed tolerance tol (for a justification of this stopping criterion, see Section 5.6).

function [x, iter]== itermeth(A,b,xO,nmax,tol,P) %ITERMETH General iterative method % X == ITERMETH(A,B,XO,NMAX,TOl,P) attempts to solve the system of % linear equations A*X==B for X. The N-by-N coefficient matrix A % must be not singular and the right hand side column vector B must % have length N. If P==' J' the Jacobi method is used, if P==' G' the % Gauss-Seidel method is selected. Otherwise, P is a N-by-N matrix % that play the role of a preconditioner. TOl specifies the tolerance % of the method. NMAX specifies the maximum number of iterations. [n, n]==size(A); if nargin ==== 6 if ischar(P)==== 1 if P====='J' I == diag(diag(A)); U == eye(n); beta == 1; alpha == 1; elseif P ==== ' G' I == tril(A);

U == eye(n); beta == 1; alpha == 1; end else [L,U]==lu(P); beta == 0; end else L == eye(n); U == L; beta == 0; end iter == 0; r == b - A * xO; rO == norm(r); err == norm (r); x == xO; while err> tol & iter < nmax iter == iter + 1; z == L\r; z == U\z; if beta ==== 0 alpha == z'*rj(z'*A*z); end x == x + alpha*z; r == b - A * x; err == norm (r) j rO; end

The Gauss-Seidel method When applying the Jacobi method, each individual component of the new vector, say x

k+l), is computed independently of the others. This may suggest that a faster convergence could be (hopefully) achieved if the new components already available xjk+l) , j = 1, . . . , i-l, together with the old ones xjk) , j > i, are used for the calculation of x

k+l) . This would lead to modifying (5.35) as follows: for k > 0 (still assuming that aii -l=- 0 for i == 1, . . . , n)

(k+l) 1 x. ==-

aii

i-l n b i - L a ijxjk+l) - L aijxjk) j

1 j

i+1

,i == 1, ..,n

(5.37)

The updating of the components is made in sequential mode, whereas in the original Jacobi method it is made simultaneously (or. in parallel). The new method, which is called the Gauss-Seidel method, corresponds to the choice P == D - E and Qk == 1, k > 0, in (5.33), where E is a lower triangular matrix whose non null entries are eij == -aij, i == 2,..., n,

j == 1, . . . , i-I. The corresponding iteration matrix is then

B == (D - E)-I(p - A).

A possible generalization is the so-called relaxation method in which p =

D - E, where w =I=- 0 is the relaxation parameter, and G-k = 1, k > 0 (see Exercise 5.13). Also for Gauss-Seidel method there exist special matrices A whose associated iteration matrices satisfy the assumptions of Proposition 5.2 (those guaranteeing convergence).

Among them let us mention: 1. matrices strictly diagonally dominant by row;

2. matrices which are symmetric and positive definite.

The Gauss-Seidel method is implemented in the Program 8 setting the input parameter P equal to ' G' . There are no general results stating that the Gauss-Seidel method converges faster than Jacobi's. However, in some special instances this is the case, as stated by the following proposition:

Proposition 5.4 Let A be a tridiagonal n x n nonsingular matrix whose diagonal elements are all nonnull. Then the Jacobi method and the Gauss- Seidel method are either both divergent or both convergent. In the latter case, the Gauss-Seidel method is faster than Jacobi's; more precisely the spectral radius of its iteration matrix is equal to the square of that of Jacobi.

Example 5.11 Let us consider a linear system Ax == b where b is chosen in such a way that the solution is the unit vector (1,1,..., l)T and A is the lOx 10 tridiagonal matrix whose diagonal entries are all equal to 3, the entries of the first lower diagonal are equal to -2 and those of the upper diagonal are all equal to -1. Both Jacobi and Gauss-Seidel methods converge since the spectral radii of their iteration matrices are strictly less than 1. More precisely, by starting from a null initial vector and setting tol =10- 12 , the Jacobi method converges in 277 iterations while only 143 iterations are requested from Gauss-Seidel's. To get this result we have used the following instructions:

» n==10; A == 3*eye(n) - 2*diag(ones(n-1,1),1) - diag(ones(n-1,1),-1); > > b==A *ones(n, 1); > > (x,iter]==itermeth(A,b,zeros(n, 1),400, 1.e-12,' J'); iter iter == 277 > > (x,iter]==itermeth(A,b,zeros(n,1),400,1.e-12, 'G'); iter iter == 143

..

....;.:..... :;:. .:::.:,," :.:

t:::: :... ... ..m .. . . :::':::;:::::::::" :"::=:::::::::::::=:

See the Exercises 5.11-5.14.

In theory iterative metllods require an infinite number of iterations to converge to the exact solution. In practice, this is neither reasonable nor necessary. Indeed we do not really need to achieve the exact solution, but rather an approximation x(k) for which we can guarantee that the error be lower than a desired tolerance E. On the other hand, since the error is itself unknown (as it depends on the exact solution), we need a suitable a posteriori error estimator which predicts the error starting from quantities that have already been computed. The first type of estimator is represented by the residual which is defined as

r(k) == b - Ax(k) ,

being x(k) the approximate value of the solution at the k-th iteration. More precisely, we could stop our iterative method at the first iteration step k min for which

llr(k rn i n) II < ellbll.

Setting x == x(k rnin) and r == r(k rnin) in (5.25) we would obtain

lle(k rn i n) II IIxll < eK(A),

which is an estimate for the relative error. We deduce that the control on the residual is meaningful only for those matrices whose condition number is reasonably small.

Example 5.12 Let us consider the linear system (5.1) where A==A 20 is the Hilbert matrix of dimension 20 introduced in Example 5.7 and b is constructed in such a way that the exact solution is x == (1,1,..., l)T. Since A is sym- metric and positive definite the Gauss-Seidel method surely converges. We use Program 8 to solve this system taking xO to be the null initial vector and set- ting a tolerance on the residual equal to 10- 5 . The method converges in 472 iterations; however the relative error is very large and equals 0.26. This is due to the fact that A is extremely ill conditioned, having K(A)

10 17 . In Figure 5.6 we show the behavior of the residual (normalized to the initial one) and that of the error as the number of iterations increases.

5.6 When should an iterative method be stopped? 129

10 1

.

10°

10- 1

10- 2 , , , 10- 3 , I " , 10- 4 \ , , ,

__ Ilr(k)/I//lr(O)/I _ /Ix - X(k) /I

10- 5

-----.. ------ --- --- --- ----.... ----..

. 100

200

. 300

. 400

500

10- 6 o

Fig. 5.6. Behavior of residual (dashed line) and error (solid line) for Gauss-Seidel iterations applied to the system of Example 5.12

An alternative approach is based on the use of a different error estima- tor, namely the increment

(krnin) = X(krnin +1) - x(krnin). More precisely, we can stop our iterative method at the first iteration step k min for which

11

(krnin)11 < Ellbll.

In the special case in which B is symmetric and positive definite, we have

lle(k) ll = lle(k+1) -

(k) ll < p(B) lle(k) ll + 1l

(k) ll.

Since p(B) should be less than 1 in order for the method to converge, we deduce

lle(k) ll < 1 11

(k) ll 1 - p(B)

(5.38)

From the last inequality we see that the control on the increment is meaningful only if p(B) is much smaller than 1 since in that case the error will be of the same size as the increment. In fact, the same conclusion holds even if B is not symmetric and positive definite (as

it occurs for the Jacobi and Gauss-Seidel methods); however in that case (5.38) is no longer true.

Example 5.13 Let us consider a system whose matrix AE IR50X50 is tridiago- nal and symmetric with entries equal to 2.001 on the main diagonal and equal to 1 on the two other diagonals. As usual, the right hand side b is chosen in such a way that the unit vector (1,..., 1)T is the exact solution. Since A is tridiagonal with strict diagonal dominance, the Gauss-Seidel method will con- verge about twice as fast as the Jacobi method (in view of Proposition 5.4). Let us use Program 8 to solve our system in which we replace the stopping criterion based on the residual by that based on the increment. Using a null

initial vector and setting the tolerance tol== 10- 5 , after 1604 iterations the program returns a solution whose error 0.0029 is quite large. The reason is that the spectral radius of the iteration matrix is equal to 0.9952, which is very close to 1. Should the diagonal entries be set equal to 3, after only 17 iterations we would have obtained an error equal to 10- 5 . In fact in that case the spectral radius of the iteration matrix would be equal to 0.428.

Let us now consider methods (5.33) for which the acceleration parame- ters ak are non-null. We call stationary the case when ak == a (a given constant) for any k > 0, dynamic the case in which ak may change along the iterations. In this framework the non singular matrix P is still called a preconditioner of A. The crucial issue is the

way the parameters are chosen. In this respect, the following result holds (see, e.g., [QV94, Chap.2], [Axe94]).

Proposition 5.5 If both P and A are symmetric and positive definite, the stationary Richardson method converges for every possible choice of x(O) iff 0 < a < 2/ Amax, where Amax (> 0) is the maximum eigenvalue of p- 1 A. Moreover, the spectral radius p(Ba) of the iteration matrix Ba == I - ap-l A is least when a == aopt, where

2 aopt == Amin + Amax

(5.39)

Amin being the minimum eigenvalue of p-l A. Under the same assumption on P and A, the dynamic Richardson method converges if for instance ak is chosen in the following way:

(Z(k))T r(k) Qk = (z(k))T Az(k)

Vk > 0

where z(k) = p- 1 r(k) is the preconditioned residual. The method (5.33) with this choice of ak is called the preconditioned gradient method, or simply the gradient method when the preconditioner P is

the identity matrix. In both cases, the following convergence estimate holds: lle(k) IIA < (

i

=

) k lle(O) IIA' k > 0,

(5.40)

where II vilA = V V T Av, \Iv ERn, is the so-called energy norm associ- ated with the matrix A.

The dynamic version should therefore be preferred to the stationary one since it does not require the knowledge of the extreme eigenvalues of p- 1 A. Rather, the parameter D:k is determined in terms of quantities which are already available from the previous iteration. We can rewrite the preconditioned gradient method more efficiently through the following algorithm (derivation is left as an exercise): given x(O), for every k > 0 do

pZ(k) == r(k) ,

(Z(k))T r(k) ak = (z(k»)T Az(k) , x(k+l) == x(k) + D:kZ(k) ,

(5.41)

r(k+l) == r(k) - D:kAz(k)

The same algorithm can be used to implement the stationary Richard- son method by simply replacing D:k with the constant value D:. From (5.'40), we deduce that if p-l A is ill conditioned the convergence rate will be very low even for D: == D:opt (as in that case p(Baopt)

1). This circumstance can be avoided provided that a convenient choice of p is made. This is the reason why P is called the preconditioner or the preconditioning matrix. If A is a generic matrix it may be a difficult task to find a precon- ditioner which guarantees an optimal trade-off between damping the condition number and keeping the computational cost for the solution of the system (5.34) reasonably low. The dynamic Richardson method is implemented in Program 8 where the input parameter P stands for the preconditioning matrix (when not prescribed, the program implements the unpreconditioned method by setting P==I).

Example 5.14 This example, of theoretical interest only, has the purpose of comparing the convergence behavior of Jacobi, Gauss-Seidel and gradient methods applied to solve the following (micro) linear system:

2XI + X2 == 1, XI + 3X2 == 0

(5.42)

with initial vector x(O) == (1, 1/2)T. Note that the system matrix is symmetric and positive definite, and that the exact solution is x == (3/5, -1/5)T. We

report in Figure 5.7 the behavior of the relative residual E(k) = Ilr(k) 1I/IIr(O) II for the three methods above. Iterations are stopped at the first iteration kmin for which E(krnin) < 10- 14 . The gradient method appears to converge the fastest.

10°

10-4

.... '.... ... \ ... \. ..... , ... , ... " ..... " ... - - Jacobi " ... ... 0 - 0 - Gauss-Seidel . ... '. ... - Gradient , ... , ... '. ... , ... , ... ". ..... , ... , ... '. ... , ... , ... \ ..... , ... , ... ". ..... , ,

10- 2

10- 6

10-8

10- 16 o

5

10 15 20 25 30 35 40

Fig. 5.7. Convergence history for Jacobi, Gauss-Seidel and gradient methods applied to system (5.42)

Example 5.15 Let us consider a system Ax == b where A E

100X100 is a pentadiagonal matrix whose main diagonal has all entries equal to 4, while the first and third lower and upper diagonals have all entries equal to -1. As customary, b is chosen in such a way that x == (1, . . . , I)T is the exact solution of our system. Let us use Program 8 which implements the preconditioned Richardson method. We fix tol=l. e-05, nmax=5000, xO=zeros (100, 1) and the preconditioner P is a tridiagonal matrix whose diagonal elements are all equal to 2, while the elements on the lower and upper diagonal are all equal to -1. Both A and P are symmetric and positive definite. The method converges in 18 iterations, whereas Program 8

(implementing the Gauss-Seidel method) requires as many as 2421 iterations before fulfilling the same stopping criterion.

Example 5.16 (Direct and iterative methods) Let us go back to Exam- ple 5.7 on the Hilbert matrix and solve the system (for different values of n) by the preconditioned gradient method using the diagonal preconditioner D, made of the diagonal entries of the Hilbert matrix. We report also the results that would be obtained using a .more efficient iterative method, the conjugate gradient method (CG). The CG method will be addressed in Section 5.8. We fix x(O) to be the null vector and iterate untill the relative residual is less than 10- 6 . In Table 5.1 we report the absolute errors (with respect to the exact solution) that are obtained with the above iterative methods and those that would be obtained by using the Gauss L U factorization approach. The error degenerates when n gets large and the direct method is used. On the other

hand, we can appreciate the benefical effect that a suitable iterative method as the CG scheme, can have on the number of iterations.

LV PG PCG n K(A) Error Error Iter. Error Iter. 4 1.55e+04 3.90e-13 1.74-02 995 2.24e-02 3 6 1.50e+07 2.62e-l0 8.80e-03 1813 9.50e-03 9 8 1.53e+ 10 1.34e-07 1.78e-02 1089 2.13e-02 4 10 1.60e+13 7.60e-04 2.52e-03 875 6.98e-03 5 12 1.67e+ 16 4.97e-Ol 1.76e-02 1355 1.12e-02 5 14 3.04e+ 17 6.65e+00 1.46e-02 1379 1.61e-02 5

Tab. 5.1. Errors obtained using the preconditioned gradient method (PG), the preconditioned conjugate gradient method (PCG) and the Gauss LV fac- torization method for the solution of the Hilbert system

See the Exercises 5.15-5.17.

.oo. 000 . .. . ", :.:.:.;.. .':':':':':.. '"':':":':':.. .:.:.:.:.:..... ... .mmm ....-..... ....

. eo . ..' .. .,:,' ,,:.'

1. An iterative method for the solution of a linear system starts from a given initial vector xeD) and builds up a sequence of vectors x(k) which we require to converge to the exact solution as k -t 00;

2. an iterative method converges for every possible choice of the initial vector xeD) iff the spectral radius of the iteration matrix is strictly less than 1;

3. classical iterative methods are those of Jacobi and Gauss-Seidel. A sufficient condition for convergence is that the system matrix be strictly diagonally dominant by row (or symmetric and definite positive in the case of Gauss-Seidel);

4. in the Richardson method convergence is accelerated thanks to the introduction of a parameter and (possibly) a convenient preconditioning matrix;

5. there are two possible stopping criteria for an iterative method: controlling the residual or controlling the increment. The former is meaningful if the system matrix is well conditioned, the latter if the spectral radius of the iteration matrix is not close to 1.

Several efficient variaIlts of the Gauss L U factorization are available for sparse systems of large dimension. Among the most advanced, we quote the so-called multifrontal method which makes use of a suitable reorder- ing of the system unknowns in order to keep the triangular factors Land U as sparse as possible. More on this issue is available on [GLS9], [DD9S] and [Iro70]. Concerning iterative methods, there exists a broad family of modern . methods, the so-called K rylov methods, which are more efficient than those presented above. Some of them feature the notable property of finite termination, that is, in exact arithmetic they provide the exact solution in a finite number of iterations. We can mention the conjugate gradient method (which can be applied if the system matrix is symmetric and positive definite) and the GMRES (Generalized Minimum RESid- ual). Their description is provided, e.g., in [Axe94] and [Saa96]. They pcg are available in the MATLAB toolbox sparfun under the name of pcg gmres and gmres. As already pointed out, iterative methods converge slowly if the system matrix is severely ill conditioned. Several preconditioning strategies have been developed (see, e.g., [dVS9]). Some of them are purely algebraic, that is, they are based on incomplete (or inexact) factorizations of the given system matrix, and are implemented in the

MATLAB functions luinc luinc or cholinc. Other strategies are developed ad hoc by exploiting cholinc the meaning and the structure of the problem which has generated the linear system at hand. Finally it is worthwhile mentioning the multigrid algorithms which are based on the sequential use of a hierarchy of systems of variable dimensions that "resemble" the original one, allowing a clever strategy of reduction of the error (see, e.g., [HacSS], [Wes91] and [Hac94]).

Exercise 5.1 For a given matrix A E IR nxn find the number of operations (as a function of n) that are needed for computing its determinant by the recursive formula (1.8).

magic Exercise 5.2 Use the MATLAB command magic (n), n==3, 4, . . . ; 500, to con- struct the magic squares of order n, that is, those matrices having entries for which the sum of the elements by rows, columns or diagonals are identical. Then compute their determinants by the command det introduced in Section

1.3 and the CPU time that is needed for this computation using the com- Inand cputime. Finally, approximate these data by the least-square method and deduce that the CPU time scales approximately as n 3 .

Exercise 5.3 Find for which values of c the matrix defined in (5.12) does not satisfy the hypotheses of Proposition 5.1. For which value

of c does this matrix become singular? Is it possible to compute the LU factorization in that case?

Exercise 5.4 Verify that the number of operations necessary to compute the LU factorization of a square matrix A of dimension n is approximately 2n 3 /3.

Exercise 5.5 Show that the LU factorization of A can be used for computing the inverse matrix A -1. (0 bserve that the j- th column vector of A -1 satisfies the linear system Ay j == ej, ej being the vector whose components are all null except the j-th component which is 1.)

Exercise 5.6 Compute the factors Land U of the matrix of Example 5.6 and verify that the L U factorization is inaccurate.

Exercise 5.7 Explain why partial pivoting by row is not convenient for sym- metric matrices.

Exercise 5.8 Consider the linear system Ax == b with

A==

2 c-2 o

-2 0 2 0 -1 3

and b such that the corresponding solution is x == (1, 1, 1) T and c is a positive real number. Compute the Gauss factorization of A and note that l32 -+ 00 when c -+ O. In spite of that, verify that the computed solution is accurate.

Exercise 5.9 Consider the linear systems Aixi == bi, i == 1,2,3, with

A 1 ==

15 6 8 11 6 6 5 3 8 5 7 6 11 3 6 9

Ai==(A1)i, i==2,3,

and b i such that the solution is always Xi == (1, 1, 1, 1) T. Solve the system by the Gauss factorization using partial pivoting by row, and comment on the obtained results.

Exercise 5.10 Show that for a symmetric and positive definite matrix A we have K(A 2) == (K(A))2.

Exercise 5.11 Analyse the convergence properties of the Jacobi and Gauss- Seidel methods for the solution of a linear system whose matrix is

ex 0 1 A== 0 ex 0 1 0 ex

ex E JR.

Exercise 5.12 Provide a sufficient condition on /3 so that both the Jacobi and Gauss-Seidel methods converge when applied for the solution of a system whose matrix is

[-10 A== /3

] .

(5.43)

Exercise 5.13 For the solution of the linear system Ax == b with A E IR nxn , consider the relaxation method: given x(O) == (xiO),...,X

O»)T, for k == 0,1,... compute

i-I n r

k) = b i - Laijx;k+l) - Laijx;k), j==1 j==i

(k) x

k+l) == (1 - W) X

k) + w

, 't 't a .. 't't

for i == 1,..., n, where w is a real parameter. Find the explicit form of the corresponding iterative matrix, then verify that the condition 0 < w < 2 is necessary for the convergence of this method. Note that if w == 1 this method reduces to the Gauss-Seidel method. If 1 < w < 2 the method is known as SO R (successive over-relaxation).

Exercise 5.14 Consider the linear system Ax = b with A = [;

] and say whether the Gauss-Seidel method converges, without explicitly computing the spectral radius of the iteration matrix.

Exercise 5.15 Compute the first iteration of the Jacobi, Gauss-Seidel and preconditioned gradient method (with preconditioner given by the diagonal of A) for the solution of system (5.42).

Exercise 5.16 Prove (5.39), then show that ( ) Amax - Amin K(P- 1 A) - 1 p Baopt = A'lnax + A'lnin = K(P-l A) + 1

(5 . 44)

Exercise 5.17 Let us consider a set of n == 20 factories which produce 20 different goods. With reference to the Leontief model introduced in Problem 5.3, suppose that the matrix C has the following integer entries: Cij == i + j - 1 for i, j == 1,..., n, while b i == i, for i == 1,..., 20. Is it possible to solve this system by the gradient method? Propose a method based on' the gradient method noting that, if A is non-singular, the matrix AT A is symmetric and positive definite.

6. Eigenvalues and eigenvectors

We consider the following problem: given a square matrix A E ffi.nxn, find a scalar A (real or complex) and a non-null vector x such that

Ax == AX.

(6.1)

Any such A is called an eigenvalue of A, while x is the associated eigen- vector. The latter is not unique; indeed all its multiples ax with a f=. 0, real or complex, are also eigenvectors associated with A. Should x be available, A can be recovered by using the Rayleigh quotient x* Ax/llxI1 2 , x* being the vector whose i-th component is equal to Xi. A number A is an eigenvalue of A if it is a root of the following poly- nomial of degree n (called the characteristic polynomial of A):

PA(A) == det(A - AI).

Consequently, a square matrix of dimension n has exactly n eigen- values (real or complex), not necessarily distinct. Also, if A has real entries, P A (A) has real coefficients, and therefore complex eigenvalues of A necessarily occur in complex conjugate pairs.

Problem 6.1 (Dynamics) Consider the system of Figure 6.1 made of two pointwise bodies PI and P 2 of mass m, connected by 2 springs and free to move along the line joining PI and P 2 . Let Xi (t) denote the position occupied by Pi at time t for i == 1,2. Then from the second law of dynamics we obtain

m XI== K(X2 - Xl) - KXI, m X2== K(XI - X2).

We are interested in forced oscillations whose corresponding solution is Xi == ai sin(wt + cP), i == 1, 2, with ai i- o. In this case we find that

-malw 2 == K(a2 - al) - Kal, -ma2w 2 == K(al - a2).

(6.2)

This is a 2 x 2 homogeneous system which has a non-trivial solution aI, a2 iff the number A == mw 2 / K is an eigenvalue of the matrix

A== [2 -1

-1] 1 .

With this definition of A, (6.2) becomes Aa == Aa. Since PA (A) == (2 - A)(l - A) - 1, the two eigenvalues are Al

2.6 18 and A2 rv 0.382 and correspond to the frequencies of oscillation Wi == vi K Ai/m which are admitted by our system. .

Xl (t)

X2 (t)

X

Fig. 6.1. The system of two pointwise bodies of equal mass connected by sprIngs

Problem 6.2 (Demography) Several mathematical models have been pro- posed in order to predict the evolution of certain species (either human or animal). The simplest population model, which was introduced in 1920 by Lotka and formalized by Leslie 20 years later, is based on the rate of mortality and fecundity for different age intervals, say i == 0, . . . , n. Let x

t) denote the number of females (males don't matter in this context) whose age at time t falls in the i-th interval. The values of x

O) are given. Moreover, let Si denote the rate of survival of the females belonging to the i-th interval, and mi the average number of females generated from a female in the i-th interval. The model by Lotka and Leslie is described by the set of equations

(t+l) (t) X i + 1 == Xi Si i == 0, . . . , n - 1, n xg+ 1) = Lx

t)mi' i==O

The first n equations describe the population development, the last its repro- duction. In matrix form we have

X(t+l) == Ax(t) ,

where x(t) == (x

t),. . . , X

))T and A is the Leslie matrix:

mo ml 80 0

m n

o

A == 0 81

o

o

o

8 n -l 0

We will see in Section 6.1 that the dynamics of this population is determined by the eigenvalue Al of A of maximum modulus, whereas the distribution of the individuals in the different age intervals (normalized with respect to the whole population), is obtained as the limit of x(t) for t ---+ 00 and satisfies Ax == AIX. This problem will be solved in Exercise 6.2. .

Problem 6.3 (Interurban viability) Let us consider n cities and let A be a matrix whose entry aij is equal to 1 if the i-th city is connected directly to the j-th city, and 0 otherwise. One can show that the components of the eigenvector x (of unit length) associated with the maximum eigenvalue pro- vides the accessibility rate (which is a measure of the ease of access) to the various cities. Consider for instance the railway network connecting the 11 main cities of Lombardy in Northern Italy, see Figure 6.2. The moduli of the components of x in this case are

0.5271 (Milan) 0.1590 (Pavia) 0.2165 (Lodi) 0.3579 (Brescia) 0.4690 (Bergamo) 0.3861 (Como) 0.1590 (Varese) 0.2837 (Lecco) 0.0856 (Sondrio) 0.1906 (Cremona) 0.0575 (Mantua)

We can see that the cities better connected are (in decreasing order) Milan, Bergamo, Como and Brescia, while Mantua is the worst connected. Note that in spite of the fact that Pavia, Varese, Mantua

and Sondrio have the same number of accesses (one), their accessibility rate is different. Obviously, this analysis does not take account at all of the frequency of the connections but merely the existence of the link between the different cities. .

In the special case where A is either diagonal or triangular, its eigenval- ues are nothing but its diagonal entries. However, if A is a general matrix and its dimension n is sufficiently large, seeking the zeros of PA('x) is not a convenient approach. Ad hoc algorithms are better suited, and one of them is described in the next section.

1 Milan 2 Pavia 3 Lodi 4 Brescia 5 Bergamo 6 Como 7 Varese 8 Lecco 9 Sondrio 10 Cremona 11 Mantua

Fig. 6.2. A schematic representation of the railway network between the main cities of Lombardy

As noticed in Problems 6.2 and 6.3, the knowledge of the whole spectrum of A (that is the the set of all its eigenvalues) is not always required. Often, only the extremal eigenvalues matter, that is, those having largest and smallest modulus. For instance, suppose that A is a square matrix of dimension n, with real entries, and assume that its eigenvalues can be ordered as

IAII> IA21 > IA31 > ... > IAnl.

(6.3)

Then, in particular, Al is distinct from the other eigenvalues of A. Let us indicate by Xl the eigenvector (with unit length) associated with Al. If the eigenvectors of A are linearly independent, Al and Xl can be computed by the following iterative procedure, commonly known as the power method: given an arbitrary initial vector x(O) and setting y(O) == x(O) / IIx(O) II, compute for k > 1

X(k) == Ay(k-I) ,

(k) _ x(k) y - IIx(k) II '

A (k) == (y(k))T Ay(k)

(6.4)

Note that, by recursion, one finds y(k) == (3(k) A ky(O) where (3(k) (ll f 111 x(i) II) -1 for k > 1. The presence of the powers of A justifies the name given to this method.

As we will see in next section, this method generates a sequence of vectors {y(k)} with unit length which, as k ----+ 00, align themselves along the direction of the eigenvector Xl. The error Ily(k) - xIII is proportional to the ratio IA2/ All in the case of a generic matrix, and to IA2/ Al12 when the matrix A is symmetric. Consequently one obtains that A (k) ----+ Al for k

00. An implementation of the power method is given in the Program 9. The iterative procedure is stopped at the first iteration k when

IA(k) - A(k-l)1 < cIA(k)l,

where E is a desired tolerance. The input parameters are the matrix A, the initial vector xO, the tolerance tol for the stopping test and the maximum admissible number of iterations nmax. Output parameters are the maximum modulus eigenvalue lambda, the associated eigenvector and the actual number of iterations which have been carried out.

fu nction [Ia m bda ,x, iter] ==eigpower(A, tol, n max,xO) %EIGPOWER Numerically evaluate one eigenvalue of a matrix. % lAMBDA == EIGPOWER(A) compute with the power method the % eigenvalue of A of maximum modulus from an initial guess which by default % is an all one vector. % lAMBDA == EIGPOWER(A,TOl,NMAX,XO) uses an absolute error % tolerance TOl (the default is 1.e-6) and a maximum number of iterations % NMAX (the default is 100), starting from the initial vector XO. % [IAM BDA, V, ITER] == EIGPOWER(A,TOl,NMAX,XO) also returns the eigenvector % V such that A*V==lAMBDA*V and the iteration number at which V was computed. [n,m] == size(A); if n -== m,

error('Only for square matrices'); end if nargin ==== 1 tol == 1.e-06; xO == ones(n, 1); nmax == 100. , end xO == xOjnorm(xO); pro == A*xO; lambda == xO'*pro; err == tol*abs(lambda) + 1; iter == 0 while err> tol*abs(lambda) & abs(lambda) -== 0 & iter <== nmax x == pro; x == xjnorm(x); pro == A*x; lambdanew == x'*pro; err == abs(lambdanew - lambda); lambda == lambdanew;

iter == iter + 1; end

Example 6.1 Consider the family of matrices Q 2 3 13 A(Q) == 5 11 10 8 Q E JR. 9 7 6 12 4 14 15 1

We want to approximate the eigenvalue with largest modulus by the power method. When Q == 30, the eigenvalues of the matrix are given by A1 == 39.396, A2 == 17.8208, A3 == -9.5022 and A4 == 0.2854 (only the first four significant digits are reported). The method approximates A1 in 22 iterations with a tolerance E == 10- 10 and x eO) == 1 T. However, if Q == -30 we need as many as 708 iterations. The different behavior can be explained by noting that in the latter case one has A1 == -30.643, A2 == 29.7359, A3 == -11.6806 and A4 == 0.5878. Thus, IA21/IA11 == 0.9704, close to unity.

Since we have assumed that the eigenvectors Xl, . . . , X n of A are linearly independent, these eigenvectors form a basis for (Cn. Thus the vectors x(O) and y(O) can be written as

n x(O) = 'L:CYiXi, i==1

n y(O) = jJ(O)'L:CYiXi, with jJ(O) = l/llx(O)II. i==1

At the first step the power method gives

n

n

X(1) = Ay(O) = jJ(O)A'L:CYiXi = jJ(O)'L:CYiAiXi i==1 i==1

and, similarly,

n y(1) = jJ(1)'L: CY i A i X i1 i==1

(1) _ 1 jJ - llx(O) II IIx(l) II .

At the generic k-th step we will have

n y(k) = ;3(k)'L:CYiA

Xi1 i==1

jJ(k) = 1 Ilx(O) Il · · · Ilx(k) Il

and therefore

y(k) = A

(3(k) (CXIXI + f ; CXi

Xi) · Since IAil All < 1 Vi == 2, . . . , n, the vector y(k) tends to align along the same direction as the eigenvector XI when k tends to +CX), provided QI =I=- O. The condition on QI, which is impossible to ensure in practice since XI is unknown, is in fact not restrictive. Actually, the effect of roundoff errors is the appearance of a non-null component along the direction of XI, even though this was not the case for the initial vector x(O). (We can say that this is one of the rare circumstances where roundoff errors help us!)

Example 6.2 Consider the matrix A(a) of Example 6.1, with a == 16. The eigenvector XI of unit length associated with AI is (1/2,1/2,1/2,

1/2)T. Let us choose (on purpose!) the initial vector (2, -2,3, -3)T, which is orthogonal to Xl. We report in Figure 6.3 the cosine of the angle contained between y(k) and Xl. We can see that after about 30 iterations of the power method the cosine tends to -1 and the angle tends to 1f, while the sequence A(k) approaches Al == 34. The power method has therefore generated, thanks to the roundoff errors, a sequence of vectors y(k) whose component along the direction of Xl is increasingly relevant.

0.2

35

o

25 """" ",:",

-0.2""

-0.8 "" """".,,,,'.

, , . ,

, ,

, .,

, .. , .. "" .

, " . ".,

- . . , , , . . , . , . . , , . . , , . . , . . , , . . , . , , . . , , ,

, . ,. , , , . .., . . .

-0.4 '

-0.6 "

-1 """""",,"" ".. ""

-1.2 o

10

20

30

40

-15 o 5 10 15 20 25 30 35 40 45

Fig. 6.3. The cosine of the angle contained between y(k) and Xl (left), and the value of A (k) (right), for k == 1, . . . ,44

Remark 6.1 It is possible to prove that the power method converges even if Al is a multiple root of PA(A). On the contrary it does not converge when there exist two distinct eigenvalues both with maximum modulus. In that case the sequence A (k) does not converge to any limit, rather it oscillates between two values.

See the Exercises 6.1-6.3.

' .

A first possible generalization of the power method consists of applying it to the inverse of the matrix A (provided A is non singular!).

Since the eigenvalues of A -1 are the reciprocals of those of A, the power method in that case allows us to approximate the eigenvalue of A of minimum modulus. In this way we obtain the so-called inverse power method: given an initial vector x(O), we set y(O) = x(O) /llx(O) II and, for k > 1, we compute

X(k) == A -ly(k-1) ,

(k) _ x(k) Y - llx(k) II'

J-L(k) == (y(k))T A -ly(k)

If A admits linearly independent eigenvectors, and if also the eigen-value An of minimum modulus is distinct from the others, then lim {L(k) = 1/ An k

oo

(i. e. (J-L (k)) -1 tends to An for k

00). At each step k we have to solve a linear system of the form Ax(k) y(k-1). It is therefore convenient to generate the LV factorization of A (or its Cholesky factorization if A is symmetric and positive definite) once for all, and then solve two triangular systems at each iteration.

Example 6.3 When applied to the matrix A(30) of Example 6.1, the inverse power method after 7 iterations yields the value 3.5037. Thus the eigenvalue of A(30) of minimum modulus will be approximately equal to 1/3.5037

0.2854. A further generalization of the power method stems from the following consideration. Let AJ-L denote the (unknown) eigenvalue of A closest to a given number (real or complex) J-L. In order to approximate AJ-L' we can at first approximate the minimum length eigenvalue, say Amin(AJ-L)' of the shifted matrix AJ-L == A - J-LI, and then set AJ-L == Amin (AIL) + J-L. We can therefore apply the inverse power method to AJ-L to obtain an approximation of Amin (AJ-L). This technique is known as the power method with shift, and the number J-L is called the shift.

In the Program 10 we implement the inverse power method with shift. By simply setting J-L == 0 we recover the inverse power method. The first 4 input parameters are the same as in Program 9, while mu is the shift. Output parameters are the eigenvalue AJ-L of A, its associated eigenvector x and the actual number of iterations that have been carried out.

function [lambda ,x, iter]==invshift(A, mu, tol, n max,xO) %INVSHIFT Numerically evaluate one eigenvalue of a matrix. % LAMBDA == INVSHIFT(A) compute with the inverse power method the % eigenvalue of A of minimum modulus. % LAMBDA == INVSHIFT(A,MU) compute the eigenvalue % of A closest to the given number (real or complex) MU. % LAMBDA == INVSHIFT(A,MU,TOL,NMAX,XO) uses an absolute error % tolerance TOL (the default is 1.e-6) and a maximum number of

iterations % NMAX (the default is 100), starting from the initial vector XO. % [LAM BDA, V, ITER] == INVSHIFT(A,MU,TOL,NMAX,XO) also returns the eigenvector % V such that A*V==LAMBDA*V and the iteration number at which V was computed. [n,m]==size(A); if n -== m, error('Only for square matrices'); end if nargin ==== 1 xO == rand(n,l); nmax == 100; tol == 1.e-06; mu == 0; elseif nargin ==== 2 xO == rand(n,l); nmax == 100; tol == 1.e-06; end [L, U]==lu (A-mu*eye(n)); if norm(xO) ==== 0 xO == rand(n,l); end xO==xO/norm(xO); zO==L \xO; pro==U\zO; lambda==xO'*pro; err==tol*a bs(lambda)+ 1; iter==O; while err> tol*abs(lambda) & abs(lambda) -== 0 & iter <== nmax x == pro; x == x/norm(x); z==L \x; pro==U\z; lambdanew == x'*pro; err == abs(lambdanew - lambda); lambda == lambdanew; iter == iter + 1; end lambda == l/lambda + mu;

Example 6.4 For the matrix A(30) of Example 6.1 we seek the eigenvalue closest to the value 17. For that we use the Program 10 with mu=17, tol =10- 10 and xO= [1; 1; 1; 1]. After 8 iterations the Program returns the value lambda=1 .2183. A less accurate knowledge of the shift would involve more

14. "::: f:::::';

f .. .."."." ". +,.49\,'

hold on/off

patch axis

iterations. For instance, if we set mu=13 the program will return the value 17.8208 after 19 iterations.

The value of the shift can be modified during the iterations, by setting J1 == A (k). This yields a faster convergence; however the computational cost grows substantially since now at each iteration the matrix AJ-l does change.

See tile Exercises 6.4-6.6.

In order to apply successfully the power method with shift we need to locate (more or less accurately) the eigenvalues of A in the complex plane. To this end let IIS introduce the following definition. Let A be a square matrix of dimension n. The Gershgorin circles ct) and Ci(c) associated with its i-th row and i-th column are respectively defined as

n ct) = {z E CC: Iz - aiil < L jaijl}, j==l,j:f:i n cy) = {z E CC: Iz - aiil < L lajil}. j==l,j:f:i

c}r) is called the i-th row circle and Ci(c) the i-th column circle. By the Program 11 we can visualize in two different windows the row circles and the column circles of a matrix. The command hold on allows the overlapping of subsequent pictures (in our case, the different circles that have been computed in sequential mode). This command can

be neutralized by the command hold off. The command patch was used in order to color the circles, while the command axis sets scaling for the x- and y-axes on the current plot.

fu nction gershci rcles(A) %GERSHCIRCLES plots the Gershgorin circles % GERSHGORINCIRCLES(A) plots the Gershgorin circles for % the square matrix A and its transpose. Abs == abs(A); n == max(size(A)); radii == sum(Abs,2)-diag(Abs); xcenter == real(diag(A)); ycenter == imag(diag(A)); theta == [O:pi/lOO:2*pi];

6.3 How to compute the shift 147

costheta == cos(theta); sintheta == sin (theta); x== []; y== []; figu re(1); clf; axis eq ua l; hold on; for i == 1: n x== [x; rad i i (i) *cos(theta) +xcenter(i)] ;y== [y; rad i i (i) *si n (theta) +ycenter(i)]; patch (x(i,:), y(i,:), , red'); end for i == l:n,
plot(x(i,:),y(i,:),'k',xcenter(i),ycenter(i),'xk'), end xmax == max(max(x)); ymax==max(max(y)); xmin == min(min(x)); ymin==min(min(y)); hold off; figure(2); clf; axis equal; hold on; radii == sum(Abs)-(diag(Abs))'; x==[]; y==[]; clf; axis equal; hold on; for i == 1: n x== [x; rad ii(i)*cos(theta)+xcenter(i)] ;y== [y; rad ii(i) *sin (theta)+ycenter(i)]; patch (x(i, :), y(i, :), , green') end for i == l:n, plot(x(i,:),y(i,:),'k',xcenter(i),ycenter(i),'xk'); end xmax == max(max(max(x)),xmax); ymax==max(max(max(y)),ymax); xmin == min(min(min(x)),xmin); ymin==min(min(min(y)),ymin); hold off; axis([xmin xmax ymin ymax]); figure(l); axis([xmin xmax ymin ymax]);

Example 6.5 In Figure 6.4 we have plotted the Gershgorin circles associated with the matrix

A==

30 1 4 15 -1 0 -3 5

2 3 -4 -2 3 5 o -1

The centers of the circles have been identified by dots.

10 5 0 -5 -10 -10 0 10 20 30

10

-10 -10

o

10

20

30

Fig. 6.4. Row circles (left) and column circles (right) for the matrix of Ex- ample 6.5

As previously anticipated, Gershgorin circles may be used to locate the eigenvalues of a matrix, as stated in the following proposition.

Proposition 6.1 All eigenvalues of a given matrix AE c nxn belong to the region of the complex plane which is the intersection of the two

regions formed respectively by the union of the row circles and the union of the column circles. Moreover, should m row circles (or column circles), with 1 < m < n, be disconnected from the union of the remaining n - m circles, then their union contains exactly m eigenvalues.

There is no guarantee that a circle should contain eigenvalues, unless it is isolated from the others. The previous result can be applied in order to obtain a preliminary guess of the shift, as we show in the following example.

Example 6.6 From the analysis of the row circles of the matrix A(30) of Example 6.1 we deduce that the real parts of the eigenvalues of A lie between -32 and 48. Thus we can use Program 10 to compute the maximum modulus eigenvalue by setting the value of the shift J-L equal to 48. The COllvergence is achieved in 16 iterations, whereas 24 iterations would be required using the power method with the same initial guess xO= [1; 1; 1; 1] and the same tolerance tol=l. a-l0.

. ',' ".

'

:;t '..:' . '. ..... ", ...;.....;. ," '.....

:::::::::'" . ...w . . .. -. .....

See the Exercises 6. 7-6.8.

Two square matrices A and B having the same dimension are called similar if there exists a non singular matrix P such that

p- 1 AP == B.

Similar matrices share the same eigenvalues. Indeed, if ,,\ is an eigenvalue of A and x -I- 0 is an associated eigenvector, we have

BP-1x == p- 1 Ax == ,,\p-1x ,

that is, ,,\ is also an eigenvalue of B and its associated eigenvector is now y == P-1x. The methods which allow a simultaneous approximation of all the eigenvalues of a matrix are generally based on the idea of transforming A (after an infinite number of steps) into a similar matrix with diagonal or triangular form, whose eigenvalues are therefore given by the entries lying on its main diagonal. Among these methods we mention the QR method which is imple- eig mented in MATLAB in the function eig. More precisely, the command D=eig(A) returns a vector D containing all the eigenvalues of A. However,

by setting [X,DJ=eig(A), we obtain two matrices: the diagonal matrix D formed by the eigenvalues of A, and a matrix X whose column vectors are the eigenvectors of A. Thus, A*X=X*D. When A is stored in a sparse mode the command eigs(A,k) allows the computation of the first k eigs eigenvalues of larger modulus of A. Should all eigenvalues of a matrix A be distinct, the sequence of eigen- values generated by the QR method converges to an upper triangular matrix similar to A, which reduces to a diagonal matrix when A is sym- metric. A description of the algorithm which stands at the basis of the QR method is provided in the Exercises 6.9 and 6.10 and, more thor- oughly, in [QSSOO], Chapter 5.

1. The power method is an iterative procedure to compute the eigenvalue of maximum modulus of a given matrix;

2. the inverse power method allows the computation of the eigenvalue of minimum modulus; it requires the factorization of the given matrix;

3. the power method with shift allows the computation of the eigenvalue closest to a given number; its effective application requires some a-priori knowledge of the location of the eigenvalues of the matrix, which can be achieved inspecting the Gershgorin circles;

4. the QR method is a global technique which allows the simultaneous approximation of all the eigenvalues of a given matrix.

See the Exercises 6.9-6.11.

:

:"."..:

:::::.?: . ... . ;:;;;:::;: ':::::::;:::;:::::'" .... >:-:-:.....:.:.:.

.:.:.>.-:-:.:.:.:-:. "

We have not analyzed the issue of the condition number of the eigen- value problem, which measures the sensitivity of the eigenvalues to the variation of the entries of the matrix. The interested reader is referred to, for instance, [Wil65], [GL89] and [QSSOO], Chapter 5. Let us just remark that the eigenvalue computation is not necessarily an ill conditioned problem when the condition number of the matrix is large. An instance of this is provided by the Hilbert matrix (see Example 5.8): although its condition number is extremely large, the eigenvalue

computation of the Hilbert matrix is well conditioned thanks to the fact that the matrix is symmetric and positive definite. Besides the QR method, for computing simultaneously all the eigenval- ues, we can use the Jacobi method which transforms a symmetric matrix into a diagonal matrix, by eliminating, step-by-step, through similarity transformations, every off-diagonal element. This method does not ter- minate in a finite number of steps since, while a new off-diagonal element is set to zero, those previously treated can reassume non-zero values. Other methods are the Lanczos method and the method which uses the so-called Sturm sequences. For a survey of all these methods see [Saa92] . The MATLAB library ARPACKC (available through the command arpackc arpackc) can be used to compute the eigenvalues of large matrices, and can be download from the address http://www.caam.rice.edu/software/ARPACK/ eigs is a command that uses this library. Let us mention that an appropriate use of the deflation technique (which consists of a successive elimination of the eigenvalues already computed) allows the acceleration of the convergence of the previous methods and hence the reduction of their computational cost.

Exercise 6.1 Upon setting the tolerance equal to E == 10- 10 , use the power method to approximate the maximum modulus eigenvalue for the following matrices, starting from the initial vector x eO) == (1,2, 3)T:

120 Al == 1 0 0 010

0.1 3.8 0 A 2 == 1 0 0 010

o -1 0 A3 == 1 0 0 010

Then comment on the convergence behavior of the method in the three differ- ent cases.

Exercise 6.2 The features of a population of fishes are described by the fol- lowing Leslie matrix introduced in Problem 6.2:

Age interval (months) x(O) mi Si 0-3 6 0 0.2 3-6 12 0.5 0.4 6-9 8 0.8 0.8 9-12 4 0.3 -

Find the vector x of the normalized distribution of this population for different age intervals, according to what we have seen in Problem 6.2.

Exercise 6.3 Prove that the power method does not converge for matrices featuring an eigenvalue of maximum modulus Al ==

eif) and another eigen- value A2 ==

e-if), where i == J=I and

, r{} E JR.

Exercise 6.4 Show that the eigenvalues of A -1 are the reciprocals of those of A.

Exercise 6.5 Verify that the power method is unable to compute the maxi- mum modulus eigenvalue of the following matrix, and explain why:

1 2 2 3 3 3 A== 1 0 -1 2 0 0 5 2 3 3 0 0 1 0

Exercise 6.6 By using the power method with shift, compute the largest positive eigenvalue and the largest negative eigenvalue of 3 1 0 0 0 0 0 1 2 1 0 0 0 0 0 1 1 1 0 0 0 A== 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 2 1 0 0 0 0 0 1 3

A is the so-called Wilkinson matrix and can be generated by the command wilkinson(7). wilkinsj

Exercise 6.7 By using the Gershgorin circles, provide an estimate of the maximum number of the complex eigenvalues of the following matrices:

A==

2 o 1 2 o

1 2 4 o o

o o 6 1

1 2 2 1 2 9

B== ,

-5 1/2 o o

o 2 1 1/4

1/2 1/2 o 1/2

1/2 o 1/2 3

Exercise 6.8 Use the result of Proposition 6.1 to find a suitable shift for the computation of the maximum modulus eigenvalue of

5 0 1 -1 0 2 0 1 A== 2 0 1 -1 1 -1 -1 0 0

Then compare the number of iterations as well the computational cost of the power method both with and without shift by setting the tolerance equal to 10- 14 .

Exercise 6.9 A matrix A E JRnxn admits a QR factorization if there exist a matrix Q E JRnxn, with the orthogonality property QTQ == I, and an upper triangular matrix R E JRnxn, such that A == QR. This factorization can be qr obtained by the command [Q, R] =qr (A). Use this command to obtain the QR factorization of the matrix A of Exercise 6.7. Then verify that the matrix C == RQ is similar to A.

Exercise 6.10 The following algorithm provides the basis for the so-called QR method for computing all the eigenvalues of a matrix A: set A (0) == A; then for k == 0,1,..., compute Q(k) and R(k) such that

A(k) == Q(k)R(k); next define A (k+l) == R (k)Q(k). Write a MATLAB program that implements this method. Then carry out a few iterations on the matrix A of Exercise 6.7. What is the limit of the sequence of matrices A (k)?

Exercise 6.11 Use the command eig to compute all the eigenvalues of the two matrices given in Exercise 6.7. Then check how accurate are the conclu- sions drawn on the basis of Proposition 6.1.

7 . Ordinary differential equations

A differential equation is an equation involving one or more derivatives of an unknown function. If all derivatives are taken with respect to a single independent variable we call it an ordinary differential equation, whereas we have a partial differential equation when partial derivatives are present. The differential equation (ordinary or partial) has order p if p the maximum order of differentiation that is present. The next chapter will be devoted to the study of a particular kind of second order partial dif- ferential equations, the elliptic equations, whereas in the present chapter we will deal with ordinary differential equations of first order. Ordinary differential equations can describe the evolution of many phenomena in various fields, as we can see from the following three ex- amples.

Problem 7.1 (Thermodynamics) Consider a body having internal tem- perature T which is set in an environment with constant temperature Te. Assume that its mass m is concentrated in a single point. Then the heat transfer between the body and the external environment can be described by the Stefan-Boltzmann law

v(t) == E

S(T4(t) - T;) where t is the time variable, E the Boltzmann constant (equal to 5.6.10- 8 Jim 2 K 4 s where J stands for Joule, K for Kelvin and, obviously, m for meter, s for sec- ond),

is the emissivity constant of the body, S the area of its surface and v is the rate of the heat transfer. The rate of variation of the energy E(t) == mCT(t) (where C denotes the specific heat of the material constituting the body) equals, in absolute value, the rate v. Consequently, setting T(O) == To, the computation of T(t) requires the solution of the ordinary differential equation

dT dt

v(t) -- mO.

(7.1)

See Exercise 7.16.

.

Problem 7.2 (Biology) Consider a population of bacteria in a confined en- vironment in which no more than B elements can coexist. Assume that, at the initial time, the number of individuals is equal to yo « B and the growth rate of the bacteria is a positive constant C. In this case the rate of change of the population is proportional to the number of existing bacteria, under the restriction that the total number cannot exceed B. This is expressed by the differential equation

dy = C y (1 _ lL) dt B '

(7.2)

whose solution y == y(t) denotes the number of bacteria at time t. Assuming that two populations Y1 and Y2 be in competition, instead of (7.2) we would have

dY1 dt = C 1 Y1 (1 - b 1 Y1 - d2Y2) , dY2 dt = -C2Y2 (1 - b 2 Y2 - d 1 Y1) ,

(7.3)

where C 1 and C 2 represent the growth rates of the two populations. The coef- ficients d 1 and d 2 govern the type of interaction between the two populations, while b1 and b 2 are related

to the available quantity of nutrients. The above equations (7.3) are called the Lotka- Volterra equations and form the basis of various applications. For their numerical solution, see Example 7.6. .

Problem 7.3 (Electricity) Consider the electrical circuit of Figure 7.1. We want to compute the function v(t) representing the potential drop at the ends of the capacitor C starting from the initial time t == 0 at which the switch I has been turned off. Assume that the inductance L can be expressed as an explicit function of the current intensity i, that is L == L(i). The Ohm law yields

_ d(i 1 L(i1)) - . R e dt -'[,1 l+ V , where R 1 is a resistance. By assuming the current fluxes to be directed as indicated in Figure 7.1, upon differentiating with respect to t both sides of the Kirchoff law i 1 == i2 + i3 and noticing that i3 == C dv / dt and i2 == v / R 2 , we find the further equation

..

di1 _ Cd2V 1 dv dt - dt 2 + R 2 dt ·

We have therefore found a system of two differential equations whose solution allows the description of the time variation of the two unknowns i1 and v. The second equation has order 2. For its solution see Example 7.7. .

L

. Z

. Z3

e

G

R 2

I

Fig. 7.1. The electrical circuit of Problem 7.3

We confine ourselves to first order differential equations, as an equation of order p > 1 can always be reduced to a system of p equations of order 1. The case of first order systems will be addressed in Section 7.8. An ordinary differential equation in general admits an infinity of so- lutions. In order to fix one of them we must

impose a further condition which prescribes the value taken by this solution at a given point of the integration interval. For instance, the equation (7.2) admits the family of solutions y(t) == B1jJ(t)j(1 + 1jJ(t)) with 1jJ(t) == e Ct + K , K being an arbitrary constant. If we impose the condition y(O) == 1, we pick up the unique solution corresponding to the value K == In[lj(B - 1)]. We will therefore consider the solllltion of the so-called Cauchy problem which takes the following form: find y : I -+ JR such that

{ y' (t) == f (t, Y (t)) Y (to) == Yo,

Vt E I,

(7.4)

where I is an interval of JR, f : I x JR -+ JR is a given function and y' denotes the derivative of y with respect to t. Finally, to is a point of I and Yo a given value which is called the initial data. In the following proposition we report a classical result of Analysis.

Proposition 7.1 Assume that the function f(t, y) is

1. continuous with respect to both arguments;

2. Lipschitz-continuous with respect to its second argument, that is, there exists a positive constant L such that

If(t, YI) - f(t, Y2)1 < LIYI - Y21, Vt E I, VYI, Y2 E JR.

Then the solution of the Cauchy problem (7.4) exists and is unique.

Unfortunately, explicit solutions are available only for very special types of ordinary differential equations. In some other cases, the solution is available only in implicit form. This is, for instance, the case with the equation y' == (y - t) / (y + t) whose solution satisfies the implicit relation 1 y 2 In(t 2 + y2) + arctg i = C,

where C is an arbitrary constant. In some other circumstances the so- lution is not even representable in implicit form, as in the case of the 2 equation y' == e- t whose general solution can only be expressed through . . a serles expansIon. For all these reasons, we seek numerical methods capable of approxi- mating the solution of every family of ordinary differential equations for which solutions do exist. The common strategy of all these methods consists of subdividing the integration interval I == [to, T], with T < +CX), into N h intervals of length h == (T - to)/Nh; h is called the discretization step. Then, at each node t n (0 < n < N h - 1) we seek the unknown value Un which approximates Yn == y(t n). The set of values {uo == YO,Ul,... ,UN h } is our numerical solution.

A classical method, the forward Euler method, generates the numerical solution as follows

U n + 1 == Un + h f n , n == 0, · · · , N h - 1

(7.5)

where we have used the shorthand notation fn == f(t n , un). This method is obtained by considering the differential equation (7.4) at every node tn, n == 1, . . . , N h and replacing the exact derivative y' (t n) by means of the incremental ratio (4.3). In a similar way, using this time the incremental ratio (4.7) to approx- imate y'(t n + 1), we obtain the backward Euler method

U n + 1 == Un + h f n+ 1 , n == 0, · · · , N h - 1

(7.6)

Both methods provide an instance of a one-step method since for com- puting the numerical solution U n +l at the node tn+l we only need the information related to the previous node tn.

More precisely, in the forward Euler method U n +l depends exclusively on the value Un previously computed, whereas in the backward Euler method it depends also on itself through the value fn+l. For this reason the first method is called the explicit Euler method and the second the implicit Euler method. For instance, the discretization of (7.2) by the forward Euler method requires at every step the simple computation of

U n +l == Un + hCu n (1 - U n / B) ,

whereas using the backward Euler method we must solve the non-linear equation

U n +l == Un + hCU n +l (1 - U n +l/ B).

Thus, implicit methods are more costly than explicit methods, since at every time-level t n + 1 we must solve a non-linear problem to compute U n +l. However, we will see that implicit methods enjoy better stability properties than explicit ones. The forward Euler method is implemented in the Program 12; the integration interval is tspan = [to, tfinal], odefun is a string which contains the function f(t, y(t)) which depends on the variables t and y, or an inline function whose first two arguments stand for t an,

y.

function [t,y]==feuler(odefun ,tspan ,y, Nh ,varargin) %FEULER Solve differential equations using the forward Euler method. % [T,Y] == FEULER(ODEFUN,TSPAN,YO,NH) with TSPAN == [TO TFINAL]

% integrates the system of differential equations y' == f(t,y) from % time TO to TFINAL with initial conditions YO using the forward Euler % method on an equispaced grid of NH intervals. % Function ODEFUN(T,Y) must return a column vector % corresponding to f(t,y). Each row in the solution array Y corresponds to % a time returned in the column vector T. % [T,Y] == FEULER(ODEFUN,TSPAN,YO,NH,Pl,P2,...) passes the additio.nal % parameters Pl,P2,... to the function ODEFUN as ODEFUN(T,Y,Pl,P2...). ,h==(tspan(2)-tspan(1)) /Nh; tt==linspace(tspa n (1), tspa n (2), N h + 1); for .

== tt(1 :end-l) y == [y; y(end,:) + h*feval(odefun,t,y(end,:),varargin{:})]; end t==tt. ,

The backward Euler method is implemented in the Program 13. Note that we have used the function fzero for the solution of the non-linear

problem at each step. As initial data for fzero we use the last computed value of the numerical solution. Note the use of the command num2str(num, s) which serves the purpose of transforming a number num in a string by keeping 16 significant digits.

fu nction [t, u]==beu ler(odefu n, tspa n,y, N h, varargin) %BEULER Solve differential equations using the backward Euler method. % [T,Y] == BEULER(ODEFUN,TSPAN,YO,NH) with TSPAN == [TO TFINAL] % integrates the system of differential equations y' == f(t,y) from % time TO to TFINAL with initial conditions YO using the backward Euler % method on an equispaced grid of NH intervals. %

Function ODEFUN(T,Y) must return a column vector % corresponding to f(t,y). Each row in the solution array Y corresponds to % a time returned in the column vector T. % [T,Y] == BEULER(ODEFUN,TSPAN,YO,NH,P1,P2,...) passes the additional % parameters P1,P2,... to the function ODEFUN as ODEFUN(T,Y,P1,P2...). h==(tspan(2)-tspan(1)) IN h; tt==li nspace(tspa n (1), tspa n (2), N h+ 1); u(l)==y; syms x; y == x; for t==tt(2:end) fun == inline(['x-' ,num2str(h,16), '*(' ,char(feval(odefun,t,y,varargin{:})),... ,) - ' , n u m 2st r (u (end) , 16)] , , x') ; u == [u, fze ro (fu n , u (end))] ; end t==tt;

A numerical method is convergent if

'In == 0, . . . , N h ,

IU n - Yn\ < C(h)

where C(h) is infinitesimal with respect to h when h tends to zero. If C(h) == O(h P) for some p > 0, then we say that the method converges with order p. In order to verify that the forward Euler method converges, we write the error as follows:

en == Un - Yn == (Un - U

) + (u

- Yn),

(7.7)

where Yn == y(t n) and

U

== Yn-I + hf(tn-I' Yn-I),

that is, u

denotes the numerical solution at time t n which we would obtain starting from the exact solution at time tn-I; see Figure 7.2. The term u

- Yn in (7.7) represents the error produced by a single step of the forward Euler method, whereas the term Un - u

represents the propagation from tn-I to t n of the error accumulated at the previous time-level tn-I. The method converges provided both terms tend to zero as h

o. Assuming that the second order derivative of Y exists and is continuous, thanks to (4.5) we find

* h 2 " (C) Un - Yn == 2 Y

n,

for a suitable

n E (tn-I, t n).

(7.8)

Un

en

U* n

hTn

Un-I - - - - - - - - - - - Yn - - - - - - - - - - - Yn-I - - - - - - - - - - -

tn-I

t n

t

Fig. 7.2. Geometrical representation of a step of the forward Euler method

Consequently, u

- Yn tends to zero as h

o. The quantity Tn(h) == (u

- Yn)/h represents the error that would be generated by forcing the exact solution to satisfy the numerical scheme, and for this reason is called the local truncation error. The (global) trun- cation error is defined as

T(h) == max ITn(h)I. n==O,..., N h

We note that these definitions of local and global truncation errors hold for a general numerical method, not only for the Euler method. In view of (7.8), the truncation error for the forward Euler method takes the following form:

T(h) == Mh/2,

(7.9)

where M == maXtE[to,T] If'(t, y(t))I. From (7.8) we deduce that limh

o T(h) == 0, and a method for which this happens is said to be consistent. Further, we say that it is consistent with order p if T(h) == O(h P) for a suitable integer p > 1. Consider now the other term in (7.7). We have

U

- Un == e n -I + h [f(t n -I' Yn-I) - f(t n -I' Un-I)] ·

(7.10)

Since f is Lipschitz continuous with respect to its second argument, we obtain

Iu

- unI < (1 + hL)Ie n -II.

If eo == 0, the previous relations yield

lenl < IU n - u

1 + lu

- Ynl < (1 + hL)le n -II + hTn(h) < [l+(l+hL)+...+(l+hL)n-l]hr(h) (1 + hL)n - 1 eL(tn-to) - 1 = L r(h) < L r(h).

We have used the identity

n-I L(l + hL)k = [(1 + hL)n - l]/hL, k==O

the inequality 1 + hL < e hL and we have observed that nh == t n - to. Therefore we find

eL(tn-to) - 1 M lenl < L T h , Vn = 0, · · · , Nh, and thus we can conclude that the forward Euler method converges with order 1. We can note that the order of this method coincides with the order of its local truncation error. This property is shared by many numerical methods for the numerical solution of ordinary differential equations.

(7.11)

Remark 7.1 The convergence estimate (7.11) is obtained by simply requiring f to be Lipschitz continuous. A better estimate, precisely le n I < Mh(t n -to)/2, holds if f satisfies the further requirement that 8f(t, y)/8y < 0 for all t E [to, T] and all -00 < y < 00. Indeed, in that case, using Tayl?r expansion, from (7.10) we obtain u

- Un = (1 + h8f /8y(t n -I, 1Jn))e n -I, thus Iu

- Un I < len-II, provided the stability inequality h < 2/ maxt 18f /8y(t, y(t))1 holds. Then lenl < Iu

- Un I + len-II < leol + nhr(h), whence we conclude owing to (7.9).

Remark 7.2 (Consistency) The property of consistency is necessary in or- der to get convergence. Actually, should it be violated, at each step the numer- ical method would generate an error which is not infinitesimal with respect to h. The accumulation with the previous errors would inhibit the global error to converge to zero when h

o.

Using a similar argument we can also prove that the backward Euler method converges with order 1 with respect to h.

Example 7.1 Consider the Cauchy problem

y'(t) = cos(2y(t)) t E (0,1], y(O) = 0,

(7.12)

whose solution is y(t) =

arcsin((e4t -1)/(e 4t + 1)). We solve it by the forward Euler method (Program 12) and the backward Euler method (Program 13). By the following commands we use different values of h: 1/2, 1/4,1/8,... ,1/512: »tspan=(O,I]; yO=O; f=inline('cos(2*y)','t','y'); > > u = i n l i n e (, 0 . 5 * a si n ((exp (4 *t)-1) . / (exp (4 *t) + 1)) , , , t ') ; » Nh=2; for k=I:10 (t,ufe]=feuler(f, tspan ,yO, Nh); fe(k)=abs(ufe(end)-feval(u, t(end))); (t,ube]=beuler(f,tspan ,yO, Nh); be(k)=abs(ube(end)-feval(u, t(end))); Nh = 2*Nh; end

The errors committed at the point t = 1 are stored in the variable fe (forward Euler) and be (backward Euler), respectively. Then we apply formula (1.11) to estimate the order of convergence. Using the following commands

> > p=log(abs(fe(1 :end-l). /fe(2:end))) /log(2); p(I:2:end) 1.2898 1.0349 1.0080 1.0019 1.0005 > > p=log(abs(be(1 :end-l). /be(2:end))) /log(2); p(1 :2:end) 0.9070 0.9720 0.9925 0.9981 0.9995

we can verify that both methods are convergent with order 1.

, Remark 7.3 The error estimate (7.11) was derived by assuming that the numerical solution {Un} is obtained in exact arithmetic. Should we account for the (inevitable) roundoff-errors, the error might blow up as h approaches o as O(l/h) (see, e.g., [Atk89]). This circumstance suggests that it might be unreasonable to go below a certain threshold h* (which is actually extremely tiny) in practical computations.

See the Exercises 7.1- 7 . 3.

"!. -

Adding together the generic steps of the forward and backward Euler methods we find the so-called Crank-Nicolson method

h U n +l = Un + 2 [In + !n+l]

(7.13)

It is a one-step implicit method, which is implemented in the Program 14. Input and output parameters are the same as in the Euler methods.

. . . ..

```
function [t,u ]==cranknic( odefun ,tspan,y, Nh ,varargin)
%CRANKNIC Solve differential equations using the Crank-Nicolson
```

method. % [T,Y] == CRANKNIC(ODEFUN,TSPAN,YO,NH) with TSPAN == [TO TFINAL] % integrates the system of differential equations y' == f(t,y) from % time TO to TFINAL with initial conditions YO using the Crank-Nicolson % method on an equispaced grid of NH intervals. % Function ODEFUN(T,Y) must return a column vector % corresponding to f(t,y). Each row in the solution array Y corresponds to % a time returned in the column vector T. % [T,Y] == CRANKNIC(ODEFUN,TSPAN,YO,NH,P1,P2,...) passes the additional % parameters P1,P2,... to the function ODEFUN as ODEFUN(T,Y,P1,P2...). h==(tspan(2)-tspan(1)) IN h; tt==linspace(tspa n (1), tspa n (2), N h+ 1); u(1)==y; fold == feval(odefun,tspan(l),y,varargin{:}); syms x; y == x; for t==tt(2:end) y == x; fun == in line(['x-' , nu m2str(h, 16),' *0.5*(' ,char(feval(odefun, t,y, varargin {: })),' +(' ,... num2str(fold, 16),'))-(' ,num2str(u(end), 16),')'], 'x'); u == [u, fzero(fun,u(end))]; fold == feval(odefun,t,u(end),varargin{:}); end t==tt. ,

The local truncation error of the Crank-Nicolson method satisfies

hTn(h)

h = y(t n) - y(tn-l) - 2 [f(t n , y(t n)) + f(tn-l, y(tn-l))]

t n = J f(t, y(t)) dt -

[J(t n , y(t n)) + f(tn-l, y(tn-l))] · tn-l

The last equality follows from the fundamental theorem of integration (which we have recalled in Section 1.4.3). The second term expresses the error associated with the trapezoidal rule for numerical integration (4.17). If we assume that y E 0 3 and use (4.18), we deduce that ( ) h 2 III ( ) Tn h = - 12 Y

n

for a suitable

n E (tn-I, t n).

(7.14)

Thus the Crank- Nicolson method is consistent with order 2, i. e. its lo- cal truncation error tends to 0 as h 2 . Using a similar approach to that followed for the forward Euler method, we can show that the Crank- Nicolson method is convergent with order 2 with respect to h.

Example 7.2 Let us solve the Cauchy problem (7.12) by using the Crank- Nicolson method with the same values of h as used in Example 7.1. As we can see, the results confirm that the error tends to zero with order 2:

> > yO==O; tspan==[O 1]; N==2; f==inline('cos(2*y)', 't' ,'y'); > > y== '0.5*asin((exp(4 *t)-1). / (exp(4 *t)+ 1))'; > > for k==1:10 [tt, u]==cranknic(f, tspan ,yO, N); t==tt(end); e(k)==abs(u(end)-eval(y));

N==2*N; end > > p==log(abs(e(1 :end-1). / e(2:end))) /log(2); p(1 :2:end) 1.7940 1.9944 1.9997 2.0000 2.0000

There is a concept of stability, called zero-stability, which guarantees that, in a fixed bounded interval, small perturbations of data yield bounded perturbations of the numerical solution when h --+ O. More precisely, a numerical method for the approximation of problem (7.4), where I == [to, T], is zero-stable if there exists 0 > 0 such that for all 8 > 0 and for any h sufficiently small

I Zn - Un I < 08, 0 < n < N h,

(7.15)

where 0 is a constant which might depend on the length of the integra- tion interval I, Zn is the solution that would be obtained by applying the

numerical method at hand to a perturbed problem, and 6 indicates the maximum size of the perturbation. Obviously, 6 must be small enough to guarantee that the perturbed problem still has a unique solution on the interval of integration. For instance, in the case of the forward Euler method, Zn satisfies { Zn+l == Zn + h [f(t n , zn) + Pn+l] , (7.16) Zo == Yo + Po for 0 < n < Nh - 1, under the assumption that IPnl < 6, 0 < n < Nh. For a consistent one-step method it can be

proved that zero-stability is a consequence of the fact that f is Lipschitz-continuous with respect to its second argument (see, e.g. [QSSOO]), in which case C depends on exp((T - to)L), where L is the Lipschitz constant. However, this is not necessarily true for other families of methods. Assume for instance that the numerical method can be written in the general form p p U n +l = Lajun-j + h Lbj!n-j + hb- 1 !n+b n = p,p + 1,. .. (7.17) j==O j==O for suitable coefficients {ak} and {b k } and p > O. This is a linear multistep method and p+ 1 denotes the number of steps. We will see some example of multistep methods in Section 7.6. The polynomial p 7r(r) = r P + 1 - Lajr P - j j==O is called the first characteristic polynomial associated with the numerical method (7.17), and we denote its roots by rj, j == 0,. . . ,p. The method (7.17) is zero-stable iff the following root condition is satisfied: { Irjl < 1 for all j == 0,... ,p; (7.18) furthermore 1[' (rj) =1= 0 for those j such that Irj I == 1. For example, for the forward Euler method we have p == 0, ao == 1, b_ 1 == 0, b o == 1. For the backward Euler method we have p == 0, ao == 1, b_ 1 == 1, b o == 0 and for the Crank-Nicolson method we have p == 0, ao == 1, b_ 1 == 1/2, b o == 1/2. In all cases there is only one root of 1[(r) which is equal to 1 and therefore all these methods are zero-stable. The following property, known as Lax- Ritchmyer equivalence theorem, is most crucial in the theory of numerical methods (see, e.g., [IK66]), and highlights the fundamental role played by the property of zero-stability:

Any consistent method is convergent iff is zero-stable.

7.5 Stability on unbounded intervals

See the Exercises 7.4-7.5.

In the previous section we considered the solution of the Cauchy problem on bounded intervals. In that context, the number N h of subintervals depends on h and can become infinite only if h goes to zero. On the other hand, there are several situations in which we wish to integrate the Cauchy problem on very large (virtually infinite) time inter- vals. In this case, even if h is fixed, Nh tends to infinity, and then results like (7.11) become meaningless as the right hand side of the inequality contains an unbounded quantity. We are interested in methods that are able to approximate the solution for arbitrarily long time-intervals, even with a step-size h relatively "large". Unfortunately, the forward Euler method, which is very cheap to im- plement, does not enjoy this property. To see this, let us consider the following model problem

{ y'(t) == Ay(t), t E (0,00), y(O) == 1,

(7.19)

where A is a negative real number. The exact solution is y(t) == eAt, which tends to 0 as t tends to infinity. Applying the forward Euler method to (7.19) we find that

Uo == 1, U n +l == u n (1 + Ah) == (1 + Ah)n+l, n > O.

(7.20)

Thus lim n -+ oo Un == 0 iff

-1 < 1 + hA < 1,

h < 2/IAI

(7.21)

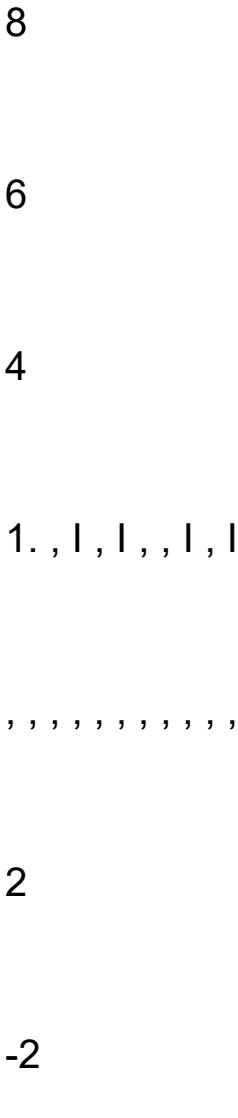
. 2.e.

This condition expresses the requirement that, for fixed h, the numeri- cal solution should reproduce the behavior of the exact solution when t n tends to infinity. If h > 2/IAI, then lim n -+ oo lunl == +00; thus (7.21) is a stability condition. The property that lim n -+ oo Un == 0 is called absolute stability.

Example 7.3 Let us apply the forward Euler method to solve problem (7.19) with A == -1. In that case we must have h < 2 for absolute stability. In Figure 7.3 we report the solutions obtained on the interval [0,30] for 3 different values

.. . . :-:'.':'; . .....:;:::.:.... '%bb«'¥&+

of h: h == 30/14(> 2), h == 30/16(< 2) and h == 1/2. We can see that in the first two cases the numerical solution oscillates. However only in the first case (which violates the stability condition) the absolute value of the numerical solution does not vanish at infinity (and actually it diverges).



, I, I I, I, I, I, I, I , I, I , I , I, I , I, I , I I I I, It .

-4

-6 o

5

10

15

20

25

30

Fig. 7.3. Solutions of problem (7.19), with A == -1, obtained by the forward Euler method, corresponding to h == 30/14(> 2) (dashed

line), h == 30/16(< 2) (solid line) and h == 1/2 (dashed-dotted line)

Similar conclusions hold when A is a negative function of t in (7.19). However in this case, IAI must be replaced by maxtE[O,(X)) IA(t)1 in the stability condition. This condition could however be relaxed to one which is less strict by using a variable step-size h n which accounts for the local behavior of IA(t)1 in every interval (tn, t n +l). In particular, the following adaptive forward Euler method could be used: choose Uo == Yo and ho == 2a/IA(to)l; then

for n == 0, 1, . . ., do

t n +l == t n + h n ,

(7.22)

U n +l == Un + hnA(tn)U n ,

hn+l == 2a/IA(t n +l)1 where a is a constant which must be less than 1 in order to have an absolutely stable method. For instance, consider the problem

y'(t) == -(10e- t + l)y(t),

t E (0,10),

with y(O) == 1. Since IA(t)1 is decreasing, the most restrictive condition for absolute stability of the forward Euler method is h < ho == 2/IA(O)1 ==

2/11. In Figure 7.4, left, we compare the solution of the forward Euler method with that of the adaptive method (7.22) for three values of Q. Note that, although every Q < 1 is admissible for stability purposes, to get an accurate solution requires choosing Q sufficiently small. In Figure 7.4, right, we also plot the behaviour of h n corresponding to the same values of Q. This picture clearly shows that the sequence {h n } increases monotonically with n.

0.04

0.7

Q == 1/3

0.6

Q == 2/3

0.5

h

0.02

,.."..""",."

,. .,." ,,,1/3 ,.". II""" .1"

0.01

Q == 1/6

--

Q == 2/3

, ,

0.1

-0.01

o o

2

4

6 t

8

10

12

0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 0.65 0.7 0.75

Fig. 7.4. Left: the numerical solution on the time interval (0.2,0.8) obtained by the forward Euler method with h == aha (dashed line) and by the adaptive variable stepping forward Euler method (7.22) (solid line) for three different values of a. Right: the behavior of the variable step-size h for the adaptive method (7.22)

In contrast to the forward Euler method, neither the backward Eu- ler method nor the Crank-Nicolson method require limitations on h for absolute stability. In fact, with the backward Euler method we obtain u n + 1 == Un + Ah U n + 1 and therefore

(1) n+ 1 U n + 1 == 1 - Ah '

which tends to zero as n ----+ 00 for all values of h > o. Similarly, with the Crank-Nicolson method we obtain

[(hA) (hA)] n+l U n +l = 1 + 2 / 1 - 2 '

which still tends to zero as n ----+ 00 for all possible values of h > o. We can conclude that the forward Euler method is conditionally abso- lutely stable, while both the backward Euler and Crank-Nicolson methods are unconditionally absolutely stable. Methods

which are unconditionally absolutely stable for all complex numbers A in (7.19) which have real

negative part are also called A-stable. Both Euler and Crank-Nicolson methods are A-stable. We point out that only implicit methods can be A-stable. This property makes implicit methods attractive in spite of being computationally more expensive than explicit methods.

Consider now the following generalized model problem

{ y'(t) == A(t)y(t) + r(t), y(O) == 1,

t E (0, +00),

(7.23)

where A and r are two continuous functions and -A max < A(t) < -Amin with 0 < Amin < Amax < +00. In this case the exact solution does not necessarily tend to zero as t tends to infinity; for instance if both rand A are constants we have

(r) At r y(t) = 1 + A e - A

whose limit when t tends to infinity is -r / A. Thus, in general, it does not make sense to require a numerical method to be absolutely stable when applied to problem (7.23). However, we are going to show that a numerical method which is absolutely stable on the model problem (7.19), if applied to the generalized problem (7.23), guarantees that the perturbations are kept under control as t tends to infinity (possibly under a suitable constraint on the time-step h). For the sake of simplicity we will confine our analysis to the forward Euler method; when applied to (7.23) it reads

{ Un+l == Un + h(AnUn + rn), Uo == 1

n > 0,

and its solution is (see Exercise 7.10)

n-l n-l n-l Un = Uo II (1 + hAk) + h Lrk II (1 + hAj), k

O k

O j

k+l

(7.24)

where Ak == A(tk) and rk == r(tk). Let us consider the following "per- turbed" method

{ Zn+l == Zn + h(AnZ n + r n + Pn+l), Zo == Uo + Po,

n > 0,

(7.25)

where Po, Pl, · · · are given perturbations which are introduced at every time step. This is a simple model in which Po and Pn+l, respectively, account for the fact that neither Uo nor r n can be determined exactly. (Should we account for all roundoff errors which are actually introduced at any step, our perturbed model would be far more involved and diffi- cult to analyze.) The solution of (7.25) reads like (7.24) provided Uk is replaced by Zk and rk by rk + Pk+l, for all k == 0, . . . , n - 1. Then

n-l n-l n-l Zn - Un = Po II (1 + hAk) + h I:Pk+l II (1 + hAj). k

O k

O j

k+I

(7.26)

It is worth noticing that this quantity does not depend on the function r(t).

i. For the sake of exposition, let us consider first the special case where >"k and Pk are two constants equal to >.. and P, respectively. Assume that h < h o (>") == 2/1>"1, which is the condition on h that ensures the absolute stability of the forward Euler method applied to the model problem (7.19). Then, using the following identity for the geometric sum

n-I I n I: k - a a - - I-a' k

O

if I a I < 1,

(7.27)

we obtain

Zn - Un = P { (1 + hA) n (1 +

) -

} .

(7.28)

It follows that the perturbation error IZn - Un I satisfies (see Exercise 7.11)

IZn - Un I < cp(>")lpl,

(7.29)

with cp(>..) == 1 if >.. < -1, while cp(>..) == 11 + 2/>"1 if -1 < >.. < 0, and }

->

Izn - Un I = I

I .

For instance, Figure 7.5 corresponds to the case where p == 0.1, >.. == -2 (left) and>" == -0.5 (right). In both cases we have taken h == h o (>")-O.OI. The conclusion that can be drawn is that the

perturbation error is bounded by Ipl times a constant which is independent of nand h. Obvi- ously, the perturbation error blows up when n increases if the stability limit h < ho is violated.

0.12 0.3

0.28 .. ..

0.1 ....:.....:.....

0.26 ., ,.." ,."., .,.,.. ,.." ,.,. .. ..."" ,..,.. :."..' ".", .,':'" ,...,. """ 0.24 ".",..... ,."....,.."....,."....." .

, ...,. ."., ."

.. ., ".. ,. ".. . . . .

. . 0.08 .....,..... . .. . ": .. .. ": .. .. . " . .. . ": .. ..

0.06 .

0.22 .. .., ... " .. . .

....:.. .. ..

0.2 '" .,. ... .,. . , '" "

. . '" ,.. , ... '" Of' ... "" '" ... . , ". ,... ... "" ... "" . . . .

0.18 ... . " .... " "... ...."..... ...: .. .." ': ....

0.04 .

. . . . . 0.16 .... , ... ,..... ..,... ....

....:. ....

0.14 ...., ... "" , ".". "" ." ."... ."... ..... "" ....

0.02 ...,... , .

. . .. "' . " . "' .,

.. """ . " ..... "'

" . " .,. . . . . - . . . . . - . . . - . - .

0.12 "".. ....." ... ... '"

.. ..

20

80

0.1 100 0

o o

40

60

20

40

60

80

100

Fig. 7.5. The perturbation error when p == 0.1: A == -2 (left) and A == -0.5 (right). In both cases h == ho(A) - 0.01

ii. In the general case where A and r are non constant, let us require h to satisfy the restriction h < ho(A), where this time ho(A) == 2/ Amax. Then,

11 + hAkl < a == a(h, Amin, Amax) == max{11 - hAm in I, 11 - hAmaxl}.

Since a < 1, we can still use the identity (7.27) in (7.26) and obtain (see Exercise 7 .11)

(n 1 - 8(h)n) lZn - Un l < Pmax a + h 1 _ c5(h) , (7.30) where Pmax == max lpkl, 8(h) == 11 - hAm in I if h < h* while 8(h) == 11 - hAm ax I if h* < h < 2/ho(A), having set h* == 2/(A m in + Amax). When h < h *, it follows that

IZn - Un I < Pmax(a n + 1/ Amin).

(7.31)

In particular,

lim IZn - Un I < Pmax/ Amin, n -+ ex:>

(7.32)

from which we still conclude that the perturbation error is bounded by Pmax times a constant which is independent of nand h (although the oscillations are no longer damped as in the previous case). In fact, similar conclusion holds also when h* < h < 2/ho(A), although this does not follow from our upper bound (7.31) which is too pessimistic in this case. In Figure 7.6 we report the perturbation errors computed on the problem (7.23), where A(t) == -2-sin(t), Pk == p(tk), p(t) == 0.1 sin(t) with h < h* (left) and with h* < h < ho(A) (right).

0.06 0.06 Pmax I Amin 0.05 - - -

0.04

0.01

0.01

o o

50

100

150

200

250

o o

50

100

150

200

250

Fig. 7.6. The perturbation error when p(t) == 0.1 sin(t) and A(t) == -2-sin(t): the step-size is h == h* - 0.01 == 0.39 (left) and h == h* + 0.01 (right)

iii. We consider now the general Cauchy problem (7.4). We claim that this problem can be related to the generalized model problem (7.23), in those cases where

-A max < af lay(t, y) < -Am in, Vt > 0, Vy E (-00, (0),

for suitable values Amin, Amax E (0, +(0). To this end, for every t in the generic interval (tn, tn+I), we subtract (7.5) from (7.16) to obtain the following equation for the perturbation error:

Zn - Un == (Zn-I - Un-I) + h{f(tn-I' Zn-I) - f(tn-I' Un-I)} + hpn.

By applying the mean value theorem we obtain

f(tn-I, Zn-I) - f(tn-I' Un-I) == An-I (Zn-I - Un-I), where An-I == fy(tn-I,

n-I), having denoted fy == aflay and

n-I being a suitable point in the interval whose endpoints are Un-I and Zn-I. Thus

Zn - Un == (1 + hAn-I)(Zn-1 - Un-I) + hpn. By a recursive application of this formula we obtain the identity (7.26), from which we derive the same conclusions drawn in ii., provided the stability restriction 0 < h < 2 I Amax holds.

Example 7.4 Let us consider the Cauchy problem

y'(t) == arctan(3y) - 3y + t, t > 0,

(7.33)

with y(O) == 1. Since fy == 3/ (1 + 9y2) - 3 is negative, we can choose Arnax == max Ify I == 3 and set h < 2/3. Thus, we can expect that the perturbations on

the forward Euler method are kept under control provided that h < 2/3. This is confirmed by the results which are reported in Figure 7.7. Note that in this example, taking h == 2/3+0.01 (thus violating the previous stability limit) the pertubation error blows up as t increases.

3.5

. . 3 ....i.....i..... .. ..i..... ..

..... - - . . . . - . . . . 2.5 . . . .

. . . . , ...

. . . .

. . . .

.. . .

. . . 2 .....;.....,...;.....;.....;..... .... - . . - .. . . - .. . .

. . 1.5 . . . . . .

. . . . , ...

. . . . . .

. . . . . .

20

40

60

80

100

Fig. 7.7. The perturbation errors when p(t) == sin(t) with h == 2/Amax - 0.01 (thick line) and h == 2/Amax + 0.01 (thin line) for the Cauchy problem (7.33)

Example 7.5 We seek a limit on h that guarantees stability for the forward Euler method applied to approximate the Cauchy problem

y' == 1 - y2, t > 0,

(7.34)

with y(O) = e - 1 . The exact solution is y(t) = (e 2 t+l - 1)j(e 2 t+l + 1) and e+1 fy == -2y. Since fy E (-2, -0.9) for all t > 0, we can take h less than ho == 1. In Figure 7.8, left, we report the solutions obtained on the interval (0,35) with h == 0.95 (thick line) and h == 1.05 (thin line). In both cases the solution oscillates, but remains bounded. Moreover in the first case, which satisfies the stability constraint, the oscillations are damped and the numerical solution tends to the exact one as t increases. In Figure 7.8, right, we report the perturbation errors corresponding to p(t) == sin(t) with h == 0.95 (thick line) and h == h* +0.1 (thin line). In both cases the perturbation errors remain bounded; moreover in the former case the upper bound (7.32) is satisfied.

In those cases where no information on y is available, finding the value Amax == max Ifyl is not a simple matter. A more heuristic approach could be pursued in these situations, by adopting a varia

le stepping procedure. Precisely, one could take t n +l == t n + h n , where

Q h n < 2 , If y (tn, Un) I

7.5 Stability on unbounded intervals 173

1.4

-.-

1.4

....

. . 1.3 .....,..,. '..'..... .. .:'.....,..,..,.

....

. . . . 1.2 ... ...,..... .:...,. . ...:.. ..,.

.....,..,..,..... .,.....,...,..... . . . . ,. . . ,. . . . . ; .. . . . . ,. .,

....

n

1.2

.....,

- . . ,,," ".....,... ....."..... .....,..... ...,.....".. . - . . . .
. . . . . - . . . . . - . . - .

1.1 . .. .. . J . A \ . ..' . ,: . ...: ...., ... 1 .... , ...
...,.. . ...;.. . ..... , ... 1 " ". . I I

I/

'

"y. - 0.8 ... ., ..,.... .\ ....., ..L 0.9 0 .. .: .\ .W . ...'..
....:.....:.....,..... '. . - . - - - -

0.8 .. . .:..

... .:. ,.. . ... ":' .... -:... .... - - . . . - . - - .

0.6 . W' ..). ... .

..

- . 0.7 ... .

... .:. .... :..... ..

. ... ':'" .... - - . 0.6 .... .,.....i.....,.....:.,.....,...,:..,..,..,.....;. .....,..,..c,.....,.."...:-.,...,.....,. . - - . - . . . . - . .'.- 0.5 ...,.....;.....,..... .:.....,.....,.;.,..,.....:.....

,.....,..,... .:....., . : : : : : : . - - - . . ; ; : : ; . 5 10 15 20 25 30 35 00

0.4 .. .' . ...

.... . 0.2 ... ...tl. :

;

0.4 a

20

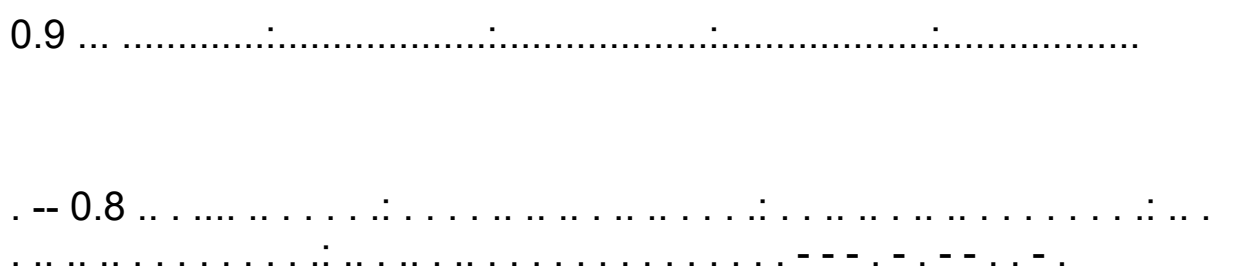
40

60

80

Fig. 7.8. On the left, numerical solutions of problem (7.34) obtained by the forward Euler method with h == 20/19 (thin line) and h == 20/21 (thick line). The values of the exact solution are indicated by circles. On the right, per- turbation errors corresponding to p(t) == sin(t) with h == 0.95 (thick line) and h == h* (thin line)

for suitable values of Q strictly less than 1. Note that the denominator depends on the value Un which is known. In Figure 7.9 we report the perturbation errors corresponding to the Example 7.5 for two different values of Q.



0.6 . .. ... ;.. ....;.. ....,.; ....- .... - - . - - - - - - - . - - - -

0.5' . ...:... ...;, ...:. ... .

0.4

0.1 ....:. .....: ....;. ..... . - - -' - - - - - .

a a

20

40

60

80

100

Fig. 7.9. The perturbation errors corresponding to p(t) == sin(t) with Q == 0.8 (thick line) and Q == 0.9 (thin line) for the Example 7.5, using the adaptive strategy

-Remark 7.4 The previous analysis can be carried out also for other kind of one-step methods, in particular for the backward Euler and Crank-Nicolson methods. For these methods which are A-stable, the same conclusions about the perturbation error can be drawn without requiring any limitation on the time-step. In fact, in the previous analysis one should replace each term 1 + hAk by (1 - hAk)-l in the backward Euler case and by (1 + hAk/2)/(1 - hAk/2) in the Crank-Nicolson case.

174 7. Ordinary differential equations

. . =...:

1. An absolutely stable method is one which generates a solution Un of the model problem (7.19) which tends to zero as t n tends to infinity;

2. a method is said A -stable if it is absolutely stable for any possible choice of the time-step h (otherwise a method is called condition- ally stable, and h should be lower than a constant depending on A);

3. when an absolutely stable method is applied to a generalized model problem (like (7.23)), the perturbation error (that is the absolute value of the difference between the perturbed and unperturbed solution) is uniformly bounded (with respect to h). In short we can say that absolutely stable method keeps the perturbation con- trolled;

4. the analysis of absolute stability for the linear model problem can be exploited to find stability conditions on the time-step when considering the nonlinear Cauchy problem (7.4) with a function f satisfying 8 f / 8y < O. In that case the stability restriction requires the step-size to be chosen as a function of 8 f / 8y. Precisely, the new integration interval [tn, t n +l] is chosen in such a way that h n == t n +l - t n satisfies h n < 2/18f(t n ,u n)/8yl.

. -

See the Exercises 7.6-7.14.

All methods presented so far are elementary examples of one-step meth- ods. More sophisticated schemes, which allow the achievement of a higher order of accuracy, are the Runge-Kutta methods and the multistep meth- ods. Runge-Kutta methods are still one-step methods; however, they in- volve several evaluations of the function f (t, y) on every interval [tn, t n + 1] . One of the most celebrated Runge-Kutta methods reads

h Un+l = Un + 6 (Kl + 2K 2 + 2K3 + K 4)

where

K I == fn, K 2 = f (t n +

, Un +

K d, K 3 = f (t n +

, Un +

K 2) , K4 == f(tn+I, Un + hK3).

It is an explicit method of order four with respect to h; at each time step, it involves four new evaluations of the function f. Other Runge-Kutta methods, either explicit or implicit, can be constructed with arbitrary order. They stand at the base of a family of MATLAB programs whose names contain the root ode followed by numbers and letters. In particular, ode ode45 is based on a pair of explicit Runge-Kutta methods (the so-called ode45 Dormand-Prince pair) of order 4 and 5, respectively. ode23 is the imple- ode23 mentation of another pair of explicit Runge-Kutta methods (the Bogacki and Shampine pair). In these methods the integration step varies in order to guarantee that the error is below a given tolerance (in these programs the default scalar relative error tolerance ReI Tol is equal to 10- 3). The program ode23tb is an implementation of an implicit Runge-Kutta for- ode23tb mula whose first stage is the trapezoidal rule, while the second stage is a backward differentiation formula of

order two (see later). Multistep methods (see (7.17)) achieve a high order of accuracy by in- volving several values Un, Un-I,. . . , for the determination of Un+I. They can be derived for instance by applying the fundamental theorem of in- tegration (which we recalled in Section 1.4.3) to the Cauchy problem (7.4), obtaining

$$t_{n+1} y_{n+1} = y_n + \int_{t_n} f(t, y(t))\, dt, \quad t_n$$

(7.35)

and then approximating the integral by a quadrature formula which involves the interpolant of f at a suitable set of nodes. A notable example of multistep method is the three-steps, third order (explicit) Adams- Bashforth formula (AB3)

$$h\, U_{n+1} = U_n + 12\,(23 f_n - 16 f_{n-1} + 5 f_{n-2})$$

(7.36)

which is obtained by replacing f in (7.35) by its interpolating polynomial of degree 2 at the nodes t_{n-2}, t_{n-1}, t_n. Another important example is the

three-steps, fourth order (implicit) Adams-Moulton formula (AM4)

h U n +l = Un + 24 (9fn+l + 19fn - Sfn-l + fn-2)

(7.37)

which is obtained by replacing f in (7.35) by its interpolating polynomial of degree 3 at the nodes t n - 2, tn-l, tn, t n + 1. Another family of multistep methods can be obtained by writing the differential equation at time t n + 1 and replacing y' (t n + 1) by a one-sided incremental ratio of high order. An instance is provided by the formula

18 9 2 6h U n +l = llu n - llu n -l + llu n -2 + 11 fn+l

(7.38)

which is known as the three-steps, third order (implicit) backward difference formula (BDF3). All these methods are consistent and zero-stable. Indeed, the first char- acteristic polynomial of both (7.36) and (7.37) is 7r(r) == r 3 - r 2 and its roots are ro == 1, rl == r2 == 0, while the first characteristic polynomial of (7.38) is 7r(r) == r 3 - 18/11r 2 + 9/11r - 2/11 and its roots are ro == 1, rl == 0.3182 + 0.2839i, r2 == 0.3182 - 0.2839i, where i is the imaginary unit. In all cases, the root condition (7.18) is satisfied. Moreover, when applied to the model

problem (7.19), AB3 is abso- lutely stable if h < 0.54/IAI while AM4 is absolutely stable if h < 3//A/. The method BDF3 is unconditionally absolutely stable for all real neg- ative A. However, this is no longer true when A E C with negative real part. In other words, BDF3 fails to be A-stable. More generally, there is no multistep A-stable method of order strictly greater than two. Multistep methods are implemented in several MATLAB programs, for ode15s instance in ode15s.

In Section 7.2 it was pointed out that implicit methods yield at each step a nonlinear problem for the unknown value U n +l. For its solution we can use one of the methods introduced in Chapter 2, or else apply the function fzero as we have done with the Programs 13 and 14. Alternatively, we can carry out fixed point iterations at every time- step. As an example, for the Crank-Nicolson method (7.13), for k

0,1, . . ., we compute until convergence

(k+l) h [(k))] u n + 1 = Un + 2 fn + f(t n +l, u n + 1 ·

It can be proved that if the initial guess U

021 is chosen conveniently, a single iteration suffices in order to obtain a numerical solution U

l whose accuracy is of the same order as the solution U n +l of the original implicit method. More precisely, if the original implicit method has order p, then the initial guess U

021 must be generated by an explicit method of order (at least) p - 1. For instance, if we use the first order (explicit) forward Euler method to initialize the Crank-Nicolson method, we get the Heun method (also called the improved Euler method), which is a second order explicit Runge- K utta method:

U

+l == Un + hin, U n +l = Un +

[fn + f(t n +l, U

+l)]

(7.39)

The explicit step is called a predictor, whereas the implicit one is called a corrector. Another example combines the (AB3) method (7.36) as predictor with the (AM4) method (7.37) as corrector. These kinds of methods are therefore called predictor-corrector methods. They enjoy the order of accuracy of the corrector method. However, being explicit, they undergo a stability restriction which is typically the same as that of the predictor method. Thus they are not adequate to integrate a Cauchy problem on unbounded intervals. In Program 15 we implement a general predictor-corrector method. The strings predictor and corrector identify the type of method that is chosen. For instance, if we use the functions eeonestep and cnonestep, which are defined in Program 16, we can call predcor as

follows: > > [t, u] ==predcor(to ,yO, T, N, f, 'eeonestep' ,'cnonestep'); and obtain the Heun method.

function [t,u]==predcor(odefun ,tspan ,y, Nh ,pred ,corr,varargin) %PREDCOR Solve differential equations using a predictor-corrector method % [T,Y]==PREDCOR(ODEFUN,TSPAN,YO,NH,PRED,CORR) with TSPAN ==

odel13

.l/t. ....:-... ," :.:..... :.....' . ..... H . . . . : .... __ . ..... '.... .... ".-

% [TO TFINAL] integrates the system of differential equations y' == f(t,y) from time % TO to TFINAL with initial conditions YO using a general predictor corrector % method on an equispaced grid of NH intervals. Function ODEFUN(T,Y) % must return a column vector corresponding to f(t,y). Each row in the % solution array Y corresponds to a time returned in the column vector T. % Functions PRED and CORR identify the type of method that is chosen. % [T,Y] == PREDCOR(ODEFUN,TSPAN,YO,NH,PRED,CORR,Pl,P2,...) passes % the additional parameters Pl,P2,... to the functions ODEFUN, PRED and % CORR as ODEFUN(T,Y,Pl,P2...), PRED(T,Y,Pl,P2...), CORR(T,Y,Pl,P2...). h==(tspa n(2)-tspan (1)) IN h; tt== [tspa n(1): h :tspan (2)]; u==y; [n,m]==size(u); if n i m, u==u';

end for t==tt(1 :end-l) y == u(:,end); fn == feval(odefun,t,y,varargin{:}); upre == feval(predictor,t,y,h,fn); ucor == feval(corrector,t+h,y,upre,h,odefun,fn,varargin{:}); u == [u, ucor]; end t == tt;

function [u]==feonestep(t,y, h ,f) u == y + h *f; return

fu nction [u]==beonestep(t, u ,y, h, f, fn, varargin) u == u + h*feval(f,t,y,varargin{:}); return

function [u]==cnonestep(t, u ,y, h, f, fn ,varargin) u == u + O.5*h* (feval(f,t,y,varargin{:})+fn); retu rn

The MATLAB program odel13 implements an Adams Moulton Bash-forth scheme with variable step-size.

See the Exercises 7.15-7.18.

Let us consider the following system of first-order ordinary differential equations whose unknowns are Y1 (t), . . . , Ym (t): Y

== f 1 (t, Y1, · · · , Ym),

Y'm == f m (t, Y1, · · · , Ym),

where t E (to, T], with the initial conditions

Y1 (to) == Yo, 1 ,

Ym(tO) == YO,m.

For its solution we could apply to each individual equation one of the methods previously introduced for a scalar problem. For instance, the n- th step of the forward Euler method would read

U r1 +1,1 == Un,l + hf1 (tn, Un, 1 , · · · , Un,m),

Un+1,m == Un,m + hf1(t n , Un, 1, ..., Un,m).

By writing the system in vector form y'(t) == F(t, y(t)), with obvious choice of notation, the extension of the methods previously developed for the case of a single equation to the vector case is straightforward. For instance, the method

Un+1 == 19F(t n +1, Un+1) + (1 - 'l9)F(t n , un),

n > 0,

with Uo == Yo, 0 < 19 < 1, is the vector form of the forward Euler method if 19 == 0, the backward Euler method if 19 == 1 and the Crank-Nicolson method if 19 == 1/2.

Example 7.6 Let us apply the forward Euler method to solve the Lotka- Volterra equations (7.3) with 0 1 == O 2 == 1, b 1 == b 2 == 0 and d 1 == d 2 == 1. In order to use Program 12 for a system of ordinary differential equations, let us create a function f which contains the component of the vector function F, which we save in the file f . m. For our specific system we have: C1==1; C2==1; d1==1; d2==1; b1==O; b2==O; yy(l)==C1 *y(l)*(l-b1 *y(l)-d2*y(2)); % first equation y(2)==-C2*y(2)*(l-b2*y(2)-d1 *y(l)); % second equation y(l)==yy(l);

Now we execute Program 12 with the following instructions

[t,u]==feuler('f' ,[0,10],[2 2],2000); plot(t, u); figure(2); plot(u(:, 1), u(: ,2));

They correspond to solving the Lotka- Volterra system on the time-interval [0, 10] with a time-step h == 0.005. The graph in Figure 7.10, left, represents the time evolution of the two components of the solution. Note that they are periodic with period 21T. The second graph in Figure 7.10, right, shows the trajectory issuing from the initial value in the so-called phase plane, that is, the cartesian plane whose coordinate axes are Yl and Y2. This trajectory is confined within a bounded region of the (Yl, Y2) plane. If we start from the point (1.2,1.2), the trajectory would stay in an even smaller region surrounding the point (1, 1). This can be explained as follows. Our differential system admits 2 points of equilibrium at which Y

== 0 and Y

== 0, and one of them is precisely (1, 1) (the other being (0, 0)) . Actually, they are obtained by solving the nonlinear system

Y

== Yl - YlY2 == 0,

Y

== -Y2 + Y2Yl == o.

If the initial data coincide with one of these points, the solution remains con- stant in time. Moreover, while (0,0) is an unstable equilibrium point, (1,1) is stable, that is, all trajectories issuing from a point near (1,1) stay bounded in the phase plane.

3

2.5 """""""""":"""

- - '. ,,,,,"""," "" .,-,.,."..., 0""""""""""" ,..,. - - - - - - . -

. - - - - - 2 "",","," "":""""""""""":""""""""""":""""""""""."""",.,""":"""""""""" - - - - -

. . . 1.5 ."""" "" ".""""""""",'.""""""",".,, ,,"""."""""'.',."""., ,. ,,"""""""."" - . - - - - . . . - . -

8

.

1 "," , , , , , , '

' , , . , , " """,' \', , , , , , , " " ".,. """""""" ." '

' , , , , , , , , , , , , , , '1 " """""""" . - . . - .- . .

0.5 0 0 0 2 4 6 8 10 0 0.5 1.5 2 2.5 3

Fig. 7.10. Numerical solutions of system (7.3). On the left, we represent Yl and Y2 on the time interval (0, 10), the solid line refers to Yl, the dashed line to Y2. Two different initial data are considered: (2,2) (thick lines) and (1.2,1.2) (thin lines). On the right, we report the corresponding trajectories in the phase plane

When we use an explicit method, the step-size h should undergo a stability restriction similar to the one encountered in Section 7.5. In

the case when the real part of the eigenvalues Ak of the Jacobian A(t) == [aFjay](t,y) ofF are all negative, we can set A == - maXt p(A(t)), where p(A(t)) is the spectral radius of A(t). This A is a candidate to replace the one entering in the stability conditions (such as, e.g., (7.21)) that were derived for the scalar Cauchy problem.

Remark 7.5 The MATLAB programs (ode23, ode45, ...) that we have men- tioned before can be used also for the solution of systems of ordinary differential equations. The synthax is odeXX (, f' , [to tf], yO), where yO is the vector of the initial conditions, f is a function to be specifed by the user and odeXX is one of the methods available in MATLAB.

Now we consider the case of an ordinary differential equation of order m

$$y^{(m)}(t) = f(t, y, y', \ldots, y^{(m-1)})$$

(7.40)

for t E (to, T], whose solution (when existing) is a family of functions de- fined up to m arbitrary constants. The latter can be fixed by prescribing m initial conditions

$$y(to) == Yo,\ y'(to) == YI,$$

$$y(m\text{-}I)(tO) == Ym.$$

Setting

$$WI(t) == y(t),\ W2(t) == y'(t),$$

$$Wm(t) == y(m\text{-}I)(t),$$

the equation (7.40) can be transformed into a first-order system of m differential equations

, wI == W2,

, W2 == W3,

, Wm-I == W m ,

W

== f(t, WI,..., W m),

with initial conditions

WI (to) == Yo, W2 (to) == YI,

Wm(to) == Ym-I.

Thus we can always approximate the solution of a differential equation of order m > 1 by resorting to the equivalent system of m first-order equations, and then applying to this system a convenient discretization method.

Example 7.7 Consider the circuit of Problem 7.3 and suppose that L(il) == L is constant and that Rl == R2 == R. In this case v can be obtained by solving the following system of two differential equations:

I V == w,

I L + RC 2 2 e w = - LC R W - LC v + LC '

(7.41)

with initial conditions v(O) == 0, w(O) == O. The system has been obtained from the second-order differential equation

d 2 v (L) dv (Rl) LC dt 2 + R 2 + Rl C dt + R 2 + 1 v = e.

We set L == 0.1 Henry, C == 10- 3 Farad, R == 10 Ohm and e == 5 Volt, where Henry, Farad, Ohm and Volt are respectively the unit measure of inductance, capacitance, resistance and voltage. Now we apply the forward Euler method with h == 0.001 seconds in the time interval [0, 0.1], by the Program 12:

> > [t,u]==feuler('fsys' ,[0,0.1],[0 0],1000);

where f sys is contained in the file f sys . m:

```
function y==fsys(t,y) L==O.l; C==1.e-03; R==10; e==5; yy(l )==y(2);
y(2)==-(L/R+R*C)/(L *C)*y(2)-2/(L *C)*y(l)+e/(L *C); y ( 1 ) ==yy ( 1 )
;
```

In Figure 7.11 we report the approximated values of v and w. As expected, v(t) tends to e/2 == 2.5 Volt for large t. In this case the real part of the eigenvalues of A(t) == [8F/8y](t,y) is negative and A can be set equal to -141.4214. Then a condition for absolute stability is to take h < 2/IAI == 0.0282.

,.:.... . .';':'.. .: '%!1

t1b

See the Exercises 7.19- 7 .20.

[-

:.

For a complete derivation of the whole family of the Runge-Kutta meth- ods we refer to [But87], [Lam91] and [QSSOO], Chapter 11. For derivation and analysis of multistep methods we refer to [Arn73]

a11d [Lam9 1] . We have not mentioned the so-called stiff problems, e.g. those asso- ciated with differential systems for which the Jacobian of F is an ill- conditioned matrix. Their numerical solution requires ad hoc implicit

7.9 What we haven't told you 183

3

180

1.5 .. ..'

80 ....,

, , ,. . """""....,.,....". ,...,.'",.., 1.1 . . .. ,'..,..".,.. h. . . . . , ...

160 "

140

120 . '

100 .

60 ...,

. . , ,. ....., """ .,,'.. """""""

1 ....,

40 ,..,,.",...,.

20, ,.....

0.5 ' , . . , .

. ,.. .,. 0 0 -20 0 0.02 0.04 0.06 0.08 0.1 0 0.02 0.04 0.06 0.08

0.1

Fig. 7.11. Numerical solution of system (7.41). The potential drop v(t) IS reported on the left, its derivative w on the right

methods, since otherwise unreasonably small time-steps would be re- quired (see [QSSOO], Chapter 11, [Lam91] and [DB02]). The MATLAB programs ode15s, ode23s and ode23tb are suited for stiff problems. Consider for instance the Van Der Pol system

I Yl == Y2, Y

== 1000(1 - yr)Y2 - Yl, Yl (0) == 2, Y2(0) == O.

(7.42)

Using the program ode45 on the interval (0,3000) would require an un- reasonably large CPU time due to the tiny time-step which is imposed by stability purpose. On the contrary, ode15s provides an accurate solution with "only" 592 time-steps (see Figure 7.12).

1 .... ,.....,...., ...,;.....,....

0.5 .

o ... "..,...."..... ."..... '. ....."...."...,.,. """" " ". ...,.....,...

-1

-0.5

-1.5 '

-2

-2.5 o

500

1000 1500 2000 2500 3000

Fig. 7.12. The first component Yl of the solution of (7.42) computed using the program ode15s. Note the non-uniform distribution of the integration nodes

We note that a variable step-size is adopted, which guarantees both stability and the desired accuracy. Its choice is based on suitable error

indicators which can be associated with the specific method at hand (see [Lam91], [SR97] and [DB02]).

Exercise 7.1 Apply the backward Euler and forward Euler methods for the solution of the Cauchy problem

$$y' = \sin(t) + y,\ t \in (0,1],\ \text{with } y(0) = 0, \tag{7.43}$$

and verify that both converge with order 1.

Exercise 7.2 Consider the Cauchy problem

$$y' = -te^{-Y},\ t \in (0,1],\ \text{with } y(0) = 0. \tag{7.44}$$

Apply the forward Euler method with h = 1/100 and estimate the number of exact significant digits of the approximate solution at t = 1

(use the property that the value of the exact solution is included between -1 and 0).

Exercise 7.3 The backward Euler method applied to problem (7.44) requires at each step the solution of the non-linear equation: U n +l = Un -ht n +l e- Un + 1 = cjJ(U n +l). The solution U n +l can be obtained by the following fixed-point it- · £ k 0 1 (k+l) A-. ((k») · h (0) P . d eratIon: or = , ,..., compute u n + 1 = '-P u n + 1 , WIt U n + 1 = Un. In under which restriction on h these iterations converge.

Exercise 7.4 Repeat Exercise 7.1 for the Crank-Nicolson method.

Exercise 7.5 Verify that the Crank-Nicolson method can be derived from the following integral form of the Cauchy problem (7.4)

y(t) - yo = r f(7, Y(7))d7 Jta

provided that the integral is approximated by the trapezoidal formula (4.1 7).

Exercise 7.6 Consider the model problem (7.19) where A is now a complex number with negative real part. Note that even in this case u(t) tends to 0 as t tends to infinity. We define the region of absolute stability of a numerical method to be the set of numbers of the complex plane of the form z = hA for which the method is absolutely stable. Determine the region of absolute stability associated with the forward Euler method.

Exercise 7.7 Solve the model problem (7.19) with A = -1+i by the forward Euler method and find the values of h for which we have absolute stability.

Exercise 7.8 Show that the Heun method defined in (7.39) is consistent. Write a MATLAB program to implement it for the solution of the Cauchy problem (7.43) and verify experimentally that the method has order of con- vergence equal to 2 with respect to h.

Exercise 7.9 Prove that the Heun method (7.39) is absolutely stable if -2 < h>'" < 0 where >... is real and negative.

Exercise 7.10 Prove formula (7.24).

Exercise 7.11 Prove the inequality (7.29).

Exercise 7.12 Prove the inequality (7.30).

Exercise 7.13 Verify the consistency of the following method (which is an explicit Runge-Kutta method of order 3)

1 U n +l = Un + 6 (k 1 + 4k 2 + k3), k 1 = hf(t n , Un), k2 == hf(t n +

, Un + k21), k3 == hf(tn+l, Un + 2k2 - k 1). Write a MATLAB program to implement it for the solution of the Cauchy problem (7.43) and verify experimentally that the method has order of convergence equal to 3 with respect to h. The methods (7.39) and (7.45) stand at the base of the MATLAB program ode23 for the solution of ordinary differen- tial equations.

(7.45)

Exercise 7.14 Prove that the method (7.45) is absolutely stable if -2.5 < h>'" < 0 where >... is real and negative.

Exercise 7.15 The modified Euler method is defined as follows:

U

+l == Un + hf(t n , Un), Un+l == Un + hf(tn+l, U

+l).

(7 . 46)

Find under which condition on h this method is absolutely stable.

Exercise 7.16 Solve equation (7.1) by the Crank-Nicolson method and the Heun method when the body in question is a cube with side equal to 1 m and mass equal to 1 Kg. Assume that To == 180K, Te == 200K,

== 0.5 and C == 100J / (Kg / K). Compare the results obtained by using h == 20 and h == 10, for t ranging from 0 to 200 seconds.

Exercise 7.17 Use MATLAB to compute the region of absolute stability of the Heun method.

Exercise 7.18 Solve the Cauchy problem (7.12) by the Heun method and verify its order.

Exercise 7.19 The displacement x(t) of a vibrating system represented by a body of a given weight and a spring, subjected to a resistive force proportional to the velocity, is described by the second-order differential equation x" +5x' + 6x == o. Solve it by the Heun method assuming that x(O) == 1 and x' (0) == 0, for t E [0, 5].

Exercise 7.20 The motion of a frictionless Foucault pendulum is described by the two equations

x" - 2w sin(w)y' + k 2 x == 0, y" + 2w cos(w)x' + k 2 y == 0,

where W is the latitude of the place where the pendulum is located, w == 7.29 . 10- 5 sec- 1 is the angular velocity of the Earth, k == y97i with 9 == 9.8 m/sec 2 and l is the length of the pendulum. Apply the forward Euler method to compute x == x(t) and y == y(t) for t ranging between 0 and 300 seconds and W == 7r/4.

8. Numerical methods for boundary-value problems

Boundary-value problems are differential problems set in an interval (a, b) of the real line or in an open multidimensional region n c :[Rd (d == 2,3) for which the value of the unknown solution (or its derivatives) is prescribed at the end-points a and b of the interval, or on the boundary an of the multidimensional region. In the multidimensional case the differential equation will involve par- tial derivatives of the exact solution with respect to the space coordi- nates. Some examples of boundary-value problems are reported below.

1. Poisson equation:

-u"(x) == f(x), x E (a, b),

(8.1)

or (in several dimensions)

-

u(x) == f(x), x E 0,

(8.2)

where f is a given function and

is the so-called Laplace operator:

d a 2 u

U=L a2 ' x. i== 1

The symbol a · / aXi denotes partial derivative with respect to the Xi variable, that is, for every point x O

au (xO) = lim u(xO + hei) - u(xO) aXi h

O h '

(8.3)

where ei is i-th unitary vector of JRd.

188 8. Numerical methods for boundary-value problems

2. H eat equation: 8u(x, t) _ 8 2 u(x, t) - f( ) 8t /-l 8x 2 - X, t ,

xE(a,b), t>O,

or (in several dimensions) 8u(x,t) 8t - /-l

u(x, t) = f(x, t), x E 0, t > 0,

where t is the time variable, J1 > 0 is a given coefficient representing the thermal conductivity, and f is again a given function. 3. Wave equation: 8 2 u(x, t) 8 2 u(x, t) 8t 2 - C 8x 2 = 0, x E (a, b), t > 0,

or (in several dimensions) 8 2 u(x, t) 8t 2 - c

u(x, t) = 0, x E 0, t > 0,

where c is a positive constant. We will restrict our attention to equations like (8.1) and (8.2). Equations depending on time, like the heat equation and the wave equation, are called initial-boundary value problems. In that case an initial condition u == Uo at t == 0 needs to be prescribed as well. For initial-boundary value problems

and for more general partial differential equations, the reader is referred for instance to [Q V94], [EEHJ 96] or [Lan03].

Problem 8.1 (Thermodynamics) If we are interested in the temperature distribution in a square [2 with side of length L, we can compute the net energy variation in each direction in an infinitesimal square with side of length l « L. We have

aJ J(x) - J(x + lei) == -le i - a (x), Xi

where J(x) represents the energy transfer for unit time. The Fourier law states that J is proportional to the variation of the temperature T, and then

a (aT) J(x) - J(x + lei) == -lei- kl- aXi aXi

= kZ 2 cPT a 2' X.

where k is a positive constant expressing the thermal conductivity coefficient. At equilibrium the sum of energy variations must vanish, and therefore d a 2 T

T(x) = 2: ox 2 (x) = 0 i== 1

x E 0,

which is a Poisson problem with f == o.

.

Problem 8.2 (Hydrogeology) The study of filtration in groundwater can lead, in some cases, to an equation like (8.2). Consider a portion n occupied by a porous medium (like ground or clay). According to the Darcy law, the water velocity filtration q == (ql, q2, q3) T is equal to the variation of the water level c/J in the medium, precisely

q == -K(8c/J/8xl, 8c/J/8x2, 8c/J/8 x 3)T,

(8.4)

where K is the constant hydraulic conductivity of the porous medium. Assume that the fluid density is constant; then the mass conservation principle yields the equation divq == 0, where divq is the divergence of the vector q and is defined as

3 d .

8qi IVq ==

8x. · . 1 2 2==

Thanks to (8.4) we therefore find that c/J satisfies the Poisson problem

c/J == o (see Exercise 8.9). .

The previous Poisson equation (8.2) admits an infinite number of solutions. With the aim of obtaining a unique solution we must impose suitable conditions on the boundary ao of O. One possibility is to prescribe the value of u on ao, that is

u(x) == g(x) for x E ao,

(8.5)

where 9 is a given function. Problem (8.2), supplemented by the boundary condition (8.5), is a Dirichlet boundary-value problem, and is precisely the problem that we will face in next section. An alternative to (8.5) is to prescribe a value for the partial derivative of u with respect to the normal direction to the boundary ao, in which case we will get a Neumann boundary-value problem. It can be proven that if f and 9 are two continuous functions and the region n is regular enough, then the Dirichlet boundary-value problem has a unique solution (while the solution of the Neumann boundary-value problem is unique up to an additive constant). In the one-dimensional case, the Dirichlet boundary-value problem takes the

following form: being given two constants a and (3 and a func- tion f == f(x), find a function u == u(x) which satisfies

-u"(x) == f(x) for x E (a, b),

u(a)==a,

u(b)=={3

(8.6)

Performing double integration it is easily seen that if f E CO ([a, b]), the solution u exists and is unique; moreover it belongs to C 2 ([a, b]). Although (8.6) is an ordinary differential problem, it cannot be cast in the form of a Cauchy problem for ordinary differential equations since the value of u is prescribed at two different points. The numerical methods which are used for its solution are based on the same principles which form the basis of the approximation of multi- dimensional boundary-value problems. This is the reason why in Section 8.1 we will make a digression on the numerical solution of problem (8.6).

We introduce on [a, b] a partition into intervals Ij == [Xj, Xj+I] for j == O,...,N with Xo == a and XN+I == b. We assume for simplicity that all intervals have the same length h.

The differential equation must be satisfied in particular at any point x j (which we call nodes from now on) internal to (a, b), that is

-u"(Xj) == f(xj), j == 1,... ,N.

We can approximate this set of N equations by replacing the second derivative with a suitable finite difference as we have done in Chapter 4 for the first derivatives. In particular, we observe that if u : [a, b] -* ffi. is a sufficiently smooth function in a neighborhood of a generic point x E (a, b), then the quantity

()2 (_) = u(x + h) - 2u(x) + u(x - h) u X h 2

(8.7)

provides an approximation to u"(x) of order 2 with respect to h (see Exercise 8.3). This suggests the use of the following approximation to problem (8.6): find {Uj } .f 1 such that _ Uj+I - 2Uj + Uj-I = f(.) h 2 XJ ,

j == 1,...,N,

(8.8)

with Uo == Q and UN+l == {3. Equations (8.8) provide a linear system AUh == h 2 f, (8.9)

where Uh == (U1,...,UN)T is the vector of unknowns, f == (f(X1) + Q / h 2 , f (X2), · · · , f (x N -1), f (x N) + /3 / h 2) T, and A is the tridiagonal ma- trix

2 -1 0 -1 2 0 -1

o

o -1 2 -1 o 0 -1 2

(8.10)

This system admits a unique solution since A is symmetric and positive definite (see Exercise 8.1). Since A is tridiagonal, this system

can be solved by the Thomas algorithm introduced in Section 5.4. We note however that, for small values of h (and thus for large values of N), A is ill-conditioned. Indeed, K(A) == Amax (A) / Amin (A) == Ch -2, for a suitable constant C independent of h (see Exercise 8.2). Consequently, the numerical solution of system (8.9), by either direct or iterative meth- ods, requires special care. In particular, when using iterative methods a suitable preconditioner ought to be employed. It is possible to prove (see, e.g., [QSSOO], Chapter 12) that if f E C 2 ([a,b]) then

. max Iu(xj) - ujI < h 2 max If II (x) I J==O,...,N+1 96xE[a,b]

that is, the finite difference method (8.8) converges with order 2 with respect to h. In Program 17 we solve the boundary-value problem

{ -u//(x) + 6u'(x) + ,u(x) == f(x) u(a) == Q

for x E (a, b),

(8.11)

u(b) == /3,

which is a generalization of problem (8.6). For this problem the finite difference method, which generalizes (8.8), reads: Uj+I - 2uj + Uj-I b

Uj+l - Uj-l . - f(.) h 2 + 2h + 1'u J - xJ'

j == 1,...,N,

Uo == Q,

UN +1 == /3.

The input parameters of Program 17 are the end-points a and b of the interval, the number N of internal nodes, the constant coefficients 6 and

1 and the string bvpfun. Finally, ua and ub represent the values that the solution should attain at x=a and x=b. Output parameters are the vector of nodes x and the computed solution uh.

function (x, uh]==bvp(a, b, N ,delta ,gamma, bvpfun, ua, ub,varargin) %BVP Solve two-point boundary value problems. % (X,UH]==BVP(A,B,N,DELTA,GAMMA,BVPFUN,UA,UB) solves with the % centered finite difference method the boundary-value % problem % -D(DU /DX)/DX+DEL TA*DU /DX+GAMMA*U==BVPFUN % on the interval (A,B) with boundary conditions U(A)==UA % and U(B)==UB. % BVPFUN can be an inline function. h == (b-a)/(N+1); z

== linspace(a,b,N+2); e == ones(N,l); A == spdiags((-e-O.5*h*delta 2*e+gamma*h A 2 -e+O.5*h*delta], -1:1, N, N); x == z(2:end-1); f == h A 2*feval(bvpfun,x,varargin{:}); f==f'; f(l) == f(l) + ua; f(end) == f(end) + ub; uh==A\f; uh==(ua; uh; ub];

x == z;

The finite element method represents an alternative to the previous finite difference method. It is derived from a suitable reformulation of problem (8.6). Let us multiply both sides of (8.6) by a generic function v. Integrating the corresponding equality on the interval (a, b), we obtain

b b - J u"(x)v(x) dx = J f(x)v(x) dx.

(8.12)

a

a

If we assume that v E C 1 ([a, b]) and use integration by parts we obtain

b b J u'(x)v'(x) dx - [u'(x)v(x)]

= J f(x)v(x) dx. a a

By making the further assumption that v vanishes at the end-points x == a and x == b, problem (8.6) becomes: find U such that u(a) == IX, u(b) == /3 and

b b J u'(x)v'(x) dx = J f(x)v(x) dx

(8.13)

a

a

for each v E C 1 ([a, b]) such that v(a) == v(b) == O. This is called weak formulation of problem (8.6). (Indeed, the test functions v can be less regular than C 1 ([a, b]), see, e.g. [QSSOO].) Its finite element approximation is defined as follows:

find Uh E Vh such that uh(a) == lX, uh(b) == /3 and

N Xj+l b

J u

(x)v

(x) dx = J f(X)Vh(X) dx,)==0 x . a J

VVh E V

(8.14)

where

Vh = {Vh E CO([a, b]): Vhllj E JIDI,j = 0,..., N},

i. e. V h is the space of continuous functions on (a, b) whose restrictions on every sub-interval lj are linear polynomials. Moreover, V

is the sub- space of Vh of those functions vanishing at the end-points a and b. Vh is called space of finite elements of degree 1.

1 ----------

C{Jk

Xk-2 Xk-1 Xk

Xk+1 Xk+2

Fig. 8.1. To the left, a generic function Vh E V

. To the right, the basis function of V

associated with the k-th node

The functions in V

are piecewise linear polynomials (see Figure 8.1, left). In particular, every function Vh of V

admits the representation

N Vh(X) = LVh(Xj)<pj(x), j==l

where

<pj(X) ==

x - Xj-l Xj - Xj-l X -Xj+l Xj - Xj+l o

if x E I j - l ,

if x E Ij,

otherwise,

for j == 1,. . . , N. Thus, 'Pk is null at every node Xj except at Xk where 'Pk(Xk) == 1 (see Figure 8.1, right). The functions 'Pj, j == 1,...,N are called shape functions and provide a basis for the vector space V

. Consequently, we can limit ourselves to fulfill (8.14) only for the shape functions <pj, j == 1, . . . , N. By exploiting the fact that 'Pj vanishes out- side the intervals I j - l and Ij, from (8.14) we obtain

J u

(x)<pj(x) dx = J f(x)<pj(x) dx, Vj = 1,..., N. (8.15) I j - 1 Ulj I j - 1 Ul j

On the other hand, we can write Uh(X) = E f 1 Uj<pj(X) + acpo(x) + (3<PN+l(X), where Uj == Uh(Xj), 'PO (x) == (a+h-x)jh for a < X < a+h, and <PN+l(X) == (x - b + h)jh for b - h < x < b, while both 'PO (x) and 'PN+l(X) are zero otherwise. By substituting this expression in (8.15), we find that for all j == 1,...,N

Uj-l J <Pj-l (x)<pj (x) dx+uj J cpj(x)cpj(x) dx lj-1 lj-1 Ul j +Uj+l J <Pj+l (x)<pj (x) dx = J f(x)<pj(x) dx + B1,j + BN,j lj l j - 1 Ul j

where

Bl,j ==

- a J cp

(x)cp

(x) dx = _ a Xl - a 10 o otherwise,

if j == 1,

while

B N,j ==

-(3 / CP

+l(X)<Pj(X) dx = - b {3 if j = N, -XN IN o otherwise.

In the special case where all intervals have the same length h, then CPj-l = -1/hinlj_1,cpj = l/h in I j - 1 and<pj = -l/hinlj,<pj+l = l/h in Ij. Consequently, we obtain for j == 1, . . . , N

-Uj-l + 2uj - Uj+l = h / f(x)cpj(x) dx + AO,j + BO,j. Ij-1Ul j

This linear system has the same matrix as the finite difference system (8.9), but a different right-hand side (and a different solution too, in spite of coincidence of notation). Finite difference and finite element solutions share however the same accuracy with respect to h when the nodal maximum error is computed. Obviously the finite element approach can be generalized to problems like (8.11) (also in the case when <5 and 1 depend on x). A further gener- alization consists of using piecewise polynomials of degree greater than 1, allowing the achievement of higher convergence orders. In these cases, the finite element matrix does not coincide anymore with that of finite differences, and the convergence order is greater than when using piece- wise linear polynomials.

See Exercises 8.1-8.8.

. .>(::. ....;; ... .:.:... ¥lt%,iPPi

Let us consider a partial differential equation, for instance equation (8.2), in a two-dimensional region !1. The idea behind finite differences relies on approximating the partial derivatives that are present in the PDE again by incremental ratios com- puted on a suitable grid (called the computational grid) made of a finite number of nodes. Then the solution u of the PDE will be approximated only at these nodes. The first step therefore consists of introducing a computational grid. Assume for simplicity that !1 is the rectangle (a, b) x (c, d). Let us in- troduce a partition of [a, b] in subintervals (Xk, Xk+l) for k == 0, . . . , N x , with Xo == a and XN x +l == b. Let us denote by

x == {xo,. . . , XN x +l} the

set of end points of such intervals and by hx == max (x k+ 1 - X k) their k

O,...,Nx maximum length. In a similar manner we introduce a discretization of the y-axis

y == {YO, · · · , YN y +l} with yo == c and YN y +l == d. The cartesian product

h ==

x X

y provides the computational grid on n (see Figure 8.2), and h == max{ h x , hy} is a characteristic measure of the grid-size. We are looking for values Ui,j which approximate U(Xi, Yj). We will assume for the sake of simplicity that the nodes be uniformly spaced, that is, Xi == xo+ihx for i == 0,..., N x +1 and Yj == yo+jhy for j == 0,..., N y +1.

Y

Y6 == d----- Y5 - - - - - - - -t - - -+- - - ;- - -

1hy

Y4 - -- Y3 - - - - - Y2 - - - - - YI Yo == c-

- - --. - -

- -

- - -

- - -

- -

- -

- - -

- - --. - -

- -

- - -

I I I - - -

- -

- -

- - -

Xo == a Xl

X2

X3 X4 == b

X

Fig. 8.2. The computational grid

h with only 15 internal nodes on a rect- angular domain

The second order partial derivatives of a function can be approximated by a suitable incremental ratio, as we did for ordinary derivatives. In the case of a function of 2 variables, we define the following incremental ratios:

82 .. = Ui-l,j - 2Ui,j + Ui+l,j 8 2 .. = Ui,j-l - 2Ui,j + Ui,j+l (8 16) x u"',J h 2 ' y u"',J h 2 · · x y

They are second order accurate with respect to hx and hy, respectively, for the approximation of 8 2 uj8x2 and 8 2 uj8y2 at the node (Xi,Yj). If we replace the second order partial derivatives of U with the formula (8.16), by requiring that the PDE is satisfied at all internal nodes of

h,

we obtain the following set of equations:

-(l5;Ui,j + l5

Ui,j) = Ii,j, i = 1,..., N x , j = 1,..., Ny.

(8.17)

We have set fi,j == f(Xi, Yj). We must add the equations that enforce the Dirichlet data at the boundary, which are

Ui,j == gi,j Vi, j such that (Xi, Yj) E O

h,

(8.18)

where OD.h indicates the set of nodes belonging to the boundary ao of O. These nodes are indicated by small squares in Figure 8.2. If we make the further assumption that the computational grid is uniform in both cartesian directions, that is, hx == hy == h, instead of (8.17) we obtain

1 - h 2 (Ui-l,j + Ui,j-l - 4Ui,j + Ui,j+l + Ui+l,j) = Ii,j, i==l,...,N x , j== 1,...,N y

(8.19)

The system given by equations (8.19) (or (8.17)) and (8.18) allows the computation of the nodal values Ui,j at all nodes of

h. For every fixed pair of indices i and j, equation (8.19) involves 5 unknown nodal values as we can see in Figure 8.3. For that reason this finite difference scheme is called the 5 point scheme for the Laplace operator. We note that the unknowns associated with the boundary nodes can be eliminated using (8.18) (or (8.17)), and therefore (8.19) involves only N == NxN y unknowns.

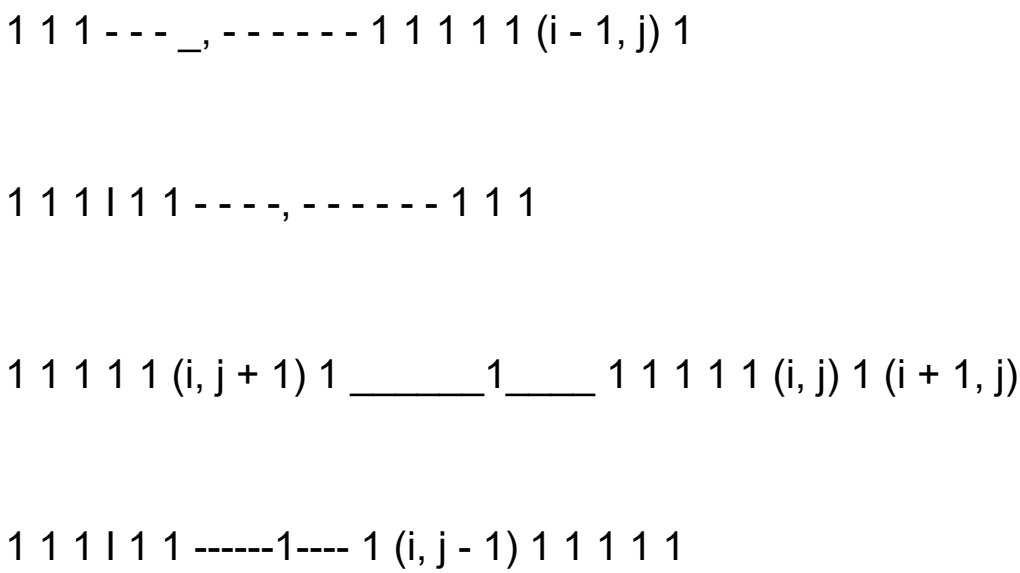


Fig. 8.3. The stencil of the 5 point scheme for the Laplace operator

The resulting system can be written in a more interesting form if we adopt the lexicographic order according to which the nodes (and, correspondingly, the unknown components) are numbered by proceeding from left to right, from the top to the bottom. We obtain a system of

the form (8.9), with a matrix A E jRNxN which takes the following block tridiagonal form:

T D 0 0 D T

A==

o

D 0

D T D o 0 D T

There are Ny rows and Ny columns, and every entry (denoted by a capital letter) consists of a N x x N x matrix. In particular, D E JR N x x N x is a diagonal matrix whose diagonal entries are -l/h

, while T E jRNxxN x is a symmetric tridiagonal matrix

2 2 1 h 2 + h 2 -- h 2 x y x 1 2 2 -- h 2 + h 2 h 2 x x y T== 0

o

o

o

1 0 -- h 2 x 1 2 2 1 -- h 2 + h 2 -- h 2 h 2 x x y x 1 2 2 0 -- h 2 + h 2 h 2 x x y

A is symmetric since all diagonal blocks are symmetric. It is also positive definite, that is v T A v > 0 \:Iv E JRN, v :/=- O. Actually, by partitioning v in Ny vectors Vi of length N x we obtain

Ny Ny-l T "T 2"T V Av =

vk TVk - h 2

Vk Vk+l. k

l y k

l

(8.20)

We can write T = 2/h

I + l/h;K where K is the (symmetric and positive definite) matrix given in (8.10). Consequently, (8.20) becomes

(V[KVl + vf Kv 2 +... + V'J.ryKVNy)/h;

which is a strictly positive real number since K is positive definite and at least one vector viis non null. Having proven that A is non-singular we can conclude that the finite difference system admits a unique solution uh.

60

0 ' . .. 10 , ... 20 .. , ... , ... , ... , ... , ... , ...

30

40

Fig. 8.4. Pattern of the matrix associated with the 5-point scheme using the lexicographic ordering of the unknowns

The matrix A is sparse since the number of non-null elements is much smaller that of null elements. (In Figure 8.4 we report the structure of the matrix corresponding to a uniform grid of 11 x 11 nodes. This picture was obtained by using the command spy (A). It

can be noted that the spy only nonzero elements lie on 5 diagonals). A sparse matrix can be stored in a special mode that allows an easy access to the nonzero elements and preserves only these elements. The command sparse that we use in the Program 18 will serve this purpose. sparse Since A is symmetric and positive definite, the associated system can be solved efficiently by either direct or iterative methods, as illustrated in Chapter 5. Finally, it is worth pointing out that A shares with its one- dimensional analog the property of being ill-conditioned: indeed, its con- dition number grows like h- 2 as h tends to zero, where h == max(h x , hy). In the Program 18 we construct and solve the system (8.17)-(8.18). The input parameters a, b, c and d denote the corners of the rectangular domain n == (a, c) x (b, d), while nx and ny denote the values of N x and Ny (the case N x i- Ny is admitted). Finally, the two strings fun and bound represent the right-hand side f == f (x, y) (otherwise called the source term) and the boundary data 9 == g(x, y). The output is a two-dimensional array u whose i, j-th entry is the nodal value Ui,j. The numerical solution can be visualized by the command mesh(x, y, u). The mesh (optional) string uex represents the exact solution of the original problem for those cases (of theoretical interest) where this solution is known. In such cases the output parameter error contains the nodal relative error

between the exact and numerical solution, which is computed as follows: error =

XIU(Xi,Yj) - Ui,jl/

xlu(Xi,Yj)l.

,J

,J

function [u ,x,y,error]==poissonfd(a ,c, b,d, nX,ny, fun, bound ,uex,varargin) %POISSONFD two-dimensional Poisson solver % [U,X,Y]==POISSONFD(A,C,B,D,NX,NY,FUN,BOUND) solves by % the 5-points finite difference scheme the problem % -LAPL(U) == FUN in the rectangle (A,C)X(B,D) with % Dirichlet boundary conditions U(X,Y)==BOUND(X,Y) for any % (X,Y) on the boundary of the rectangle. % % [U ,X,Y, ERROR]==POISSON FD(A, C, B, D, NX, NY, FU N, BOU N D, U EX) computes % also the maximum nodal error ERROR with respect to the exact % solution UEX. % FUN, BOUND and UEX can be online functions. if nargin ==== 8 u ex == i n l i n e (, 0' , , x' " y') ; end nx==nx+l; ny==ny+l; hx == (b-a)/nx; hy == (d-c)/ny; nxi == nx+l; hx2 == hXA2; hy2 == hyA2; kii == 2/hx2+2/hy2; kix == -1/hx2; kiy == -1/hy2; dim == (nx+l)*(ny+l); K == speye(dim,dim); rhs == zeros(dim,l); y == c; for m == 2:ny x == a; y == y + hy; for n == 2:nx i == n+(m-l)*(nx+l); x == x + hx; rhs(i) == feval(fun,x,y,varargin{:}; K(i,i) == kii; K(i , i-l) == kix; K(i,i+l) == kix; K(i,i+nxl) == kiy; K(i,i-nxl) == kiy; end end rhsl == zeros(dim,l);

x == [a:hx:b]; rhs1(1:nx1) == feval(bound,x,c,varargin{:}); rhs1(dim-nx:dim) == feval(bound,x,d ,varargin{:}); y == [c:hy:d]; rhs1(1:nx1:dim-nx) == feval(bound,a,y,varargin{:}); rhs1(nx1:nx1:dim) == feval(bound,b,y,varargin{:}); rhs == rhs - K*rhs1; nbound == [[1: nx1], [dim-nx:dim], [1: nx1 :dim-nx], [nx1: nx1 :dim]]; ninternal == setdiff([l:dim],nbound); K ==

K(ninternal,ninternal); rhs == rhs(ninternal); utemp == K\rhs; uh == rhs1; uh (ninternal) == utemp; k == 1; y == c; for j == 1:ny+1

x == a; for i == 1:nx1 u (i ,j) == u h (k) ; k == k + 1; ue(i,j) == feval(uex,x,y,varargin{:}); x == x + hx; end y == y + hy; end x == [a:hx:b]; y == [c: hy:d]; if nargout ==== 4 & nargin ==== 8 warning('Exact solution not available'); error == [ ]; else error == max(max(abs(u-ue)))/max(max(abs(ue))); end, end retu rn

Example 8.1 The transverse displacement u of an elastic membrane from a reference plane 0 == (0,1)2 under a load whose intensity is f(x, y) == 87r 2 sin(27rx) cos(27rY) satisfies a Poisson problem like (8.2) in the domain O. The Dirichlet value of the displacement is prescribed on 80 as follows: 9 == 0 on the sides x == 0 and x == 1, and g(x,O) == g(x, 1) == sin(27rx), 0 < x < 1. This problem admits the exact solution u(x,y) == sin(27rx)cos(27rY). In Figure 8.5 we show the numerical solution obtained by the five-point finite difference scheme on a uniform grid. Two different values of h have been used: h == 1/10 (left) and h == 1/20 (right). When h decreases the numerical solution improves, and ac- tually the nodal relative error is 0.0292 for h == 1/10 and 0.0081 for h == 1/20. .

I 1 1 1 M.

Mi 0.:.

;1<

:;::3

r;1

;#{

::s

",. 0.2 . "";'::;-"" 0.8 o 1

i

{

AO Y'- 0.6

-0.5 -{

1 . .. - ...". .-. .:

.. . . . _ :. : .. (. (. : ,

. . , . ' . " ; . f . > . { . ': . ' . : . _. . : . : .

. . . f .

. ' . .. . _ .

: . : . ; o ; . . . ' . : .

o -:3 .

2 .::;)::;

.. 0 -1 L

=

;t;tets,

o

j:

.;> :'-'". o 1

Fig. 8.5. Transverse displacement of an elastic membrane computed on two uniform grids. On the horizontal plane we report the isolines of the numerical solution. The triangular partition of 0 only serves the purpose of the visual- ization of the results

Also the finite element method can be easily extended to the two-dimensional case. To this end the problem (8.2) must be

reformulated in an integral form and the partition of the interval (a, b) in one dimension must be replaced by a decomposition of n by polygons (typically, trian- gles) called elements. The shape function <{Jk will still be a continuous function, whose restriction on each element is a polynomial of degree 1 on each element, which is equal to 1 at the k-th vertex (or node) of the triangulation and 0 at all other vertices. For its implementation one can pde use the MATLAB toolbox pde.

In the previous section we have shown that the solution of the finite difference problem exists and is unique. Now we investigate the approx- imation error. We will assume for simplicity that hx == hy == h. If

8:XIU(Xi, Yj) - ui,jl

0 as h

0,

,J

the method is called convergent. As we have already pointed out, consistency is a necessary condition for convergence. A method is consistent if the residual that is obtained when the exact solution is plugged into the numerical scheme tends to zero when h tends to zero. If we consider the five point finite difference scheme, at every internal node (Xi, Yj) of

h we define

Th(Xi, Yj) == - f(Xi, Yj) U(Xi-l, Yj) + U(Xi, Yj-l) - 4U(Xi, Yj) + U(Xi, Yj+l) + U(Xi+l, Yj) h 2

This is the local truncation error at the node (Xi, Yj). By (8.2) we obtain

( ) { 82u (_ _) _ U(Xi-l, Yj) - 2U(Xi, Yj) + U(Xi+l' Yj) } Th Xi,Yj = ax2 xZ'YJ h 2 { 82u (_ _) _ U(Xi, Yj-l) - 2U(Xi, Yj) + U(Xi, Yj+l) } + ay2 Xz, YJ h 2 ·

Thanks to the analysis that was carried out in Section 8.2 we can con- clude that both terms vanish as h tends to o. Thus

lim Th(Xi, Yj) == 0, V(Xi, Yj) E

h \ 8

h, h

O

that is, the five-point method is consistent. It is also convergent, as stated in the following Proposition (for its proof, see, e.g., [IK66]):

Proposition 8.1 Assume that the exact solution u E C 4 (0), i.e. all its partial derivatives up to the 4th order are continuous in the closed domain n. Then there exists a positive constant C such that

max I u (x- Y o) - u- _ I < CMh 2 - -

, J

,J -

,J

where M is the maximum absolute value attained by the fourth order derivatives of u in n.

Example 8.2 Let us verify that the 5 point scheme applied to solve the Pois- son problem of Example 8.1 converges with order 2 with respect to h. We start from h == 1/4 and, then we halve subsequently the value of h, until h == 1/64, through the following instructions:

> > a==O; b== 1 ;c==O;d== 1; > > f==inline('8*pi"'2*sin(2*pi*x). *cos(2*pi*y)', 'x', 'y'); > > g==inline('sin(2*pi*x). *cos(2*pi*y)', 'x', 'y'); uex==g; nx==4; ny==4; > > for n==1:5 [u,x,y,error(n)]==poissonfd(a,c,b,d,nx,ny,f,g,uex); nx == 2*nx; ny == 2*ny; end

The vector containing the error is > > format short e; error 1.3565e-01 4.3393e-02 1.2308e-02 3.2775e-03 8.4557e-04

As we can verify using the following commands

> > log(abs(error(1:end-1). / error(2:end))) /log(2) 1.6443e+OO 1.817ge+OO 1.908ge+OO 1.9546e+OO

this error decreases as h 2 .

#I

1. Boundary-value problems are differential equations set in a spatial domain n c JRd (which is an interval if d == 1) that require information on the solution on the domain boundary; 2. finite difference approximations are based on the discretization of the given differential equation at selected points (called nodes) where derivatives are replaced by finite difference formulae;

3. the finite difference method provides a nodal vector whose components converge to the corresponding nodal values of the exact solution quadratically with respect to the grid-size; 4. the finite element method is based on a suitable integral reformu- lation of the original differential equation, then on the assumption that the approximate solution is a piecewise polynomial; 5. matrices arising

from both finite difference and finite element ap- proximations are sparse and ill-conditioned.

"IJ! -

See Exercises 8.9-8.10.

We could simply say that we have told you almost nothing, since the field of numerical analysis which is devoted to the numerical approximation of partial differential equations is so broad and multifaceted to deserve an entire monograph simply for addressing the most essential concepts (see, e.g., [TW98], [EEHJ96]). We would like to mention that the finite element method is nowadays probably the most widely diffused method for the numerical solution of partial differential equations (see, e.g., [QV94], [Bra97], [BS01]). As already mentioned the MATLAB toolbox pdetool allows the solution of a broad family of partial differential equations by the linear finite element method. Other popular techniques are the spectral methods (see, [CHQZ88], [Fun92], [BM92], [KS99]) and the finite volume method (see, [Kro98], [Hir88] and [LeV02]).

Exercise 8.1 Verify that matrix (8.10) is positive definite.

Exercise 8.2 Verify that the eigenvalues of the matrix (8.10) are

A j == 2 (1 - cos (j 0)) , j == 1, . . . , N,

while the corresponding eigenvectors are

Qj == (sin(jO), sin(2jO), . . . , sin(njO)) T,

where () == 7r/(n + 1). Deduce that K(A) is proportional to h- 2 .

Exercise 8.3 Prove that the quantity (8.7) provides a second order approxi- mation of u" (x) with respect to h.

Exercise 8.4 Compute the matrix and the right hand side of the numerical scheme that we have proposed to approximate problem (8.11).

Exercise 8.5 Use the finite difference method to approximate the boundary- value problem

" k w ( ) -u + T u = T in 0, 1 , u(O) == u(l) == 0,

where u == u(x) represents the vertical displacement of a string of length 1, subject to a transverse load of intensity w per unit length. T is the tension and k is the elastic coefficient of the string. For the case in which w == 1 + sin(47rx), T == 1 and k == 0.1, compute the solution corresponding to h == l/i, i == 10,20,40, and deduce the order of accuracy of the method.

Exercise 8.6 We consider problem (8.11) on the interval (0,1) with 'Y == 0, f == 0, Q == 0 and {3 == 1. Using the Program 17 find the maximum value h crit of h for which the numerical solution is monotone (as is the exact solution) when 8 == 100. What happens if 8 == 1000? Suggest an empirical formula for h crit (8) as a function of 8, and verify it for several values of 8.

Exercise 8.7 Use the finite difference method to solve problem (8.11) in the case where the following Neumann boundary conditions are prescribed at the endpoints

u' (a) == Q, U' (b) == (3.

Use the formulae given in (4.10) to discretize u' (a) and u' (b) .

Exercise 8.8 Verify that, when using a uniform grid, the right hand side of the system associated with the centered finite difference scheme coincides with that of the finite element scheme provided

that the composite trapezoidal formula is used to compute the integrals on the elements I k - 1 and Ik.

Exercise 8.9 Verify that div\l <P ==

<P, where \l is the gradient operator that associates to a function u the vector whose components are the first order partial derivatives of u.

Exercise 8.10 Consider a square plate whose side is 20 cm and thermal con- ductivity is k == 0.2 cal/sec.cm.C. Denote by Q == 5 cal/cm 3 .sec the heat production rate per unit area. The temperature T == T(x, y) of the plate sat- isfies tile equation -

T == Q/k. Assuming that T is null on three sides of the plate and is equal to 1 on the fourth side, determine the temperature T at the center of the plate.

9. Solutions of the exercises

Solution 1.1 Only the numbers of the form ::i:0.1al . 2 t with al == 0,1 and t == ::i:2, ::i:1, 0 belong to the set JF(2, 2, -2,2). For a given exponent, we can represent in this set only the two numbers 0.10 and 0.11, and their opposites. Consequently, the number of elements belonging to JF(2, 2, -2,2) is 20. Finally, EM == 1/2.

Solution 1.2 For any fixed exponent, each of the digits a2, . . . , at can assume 13 different values, while al can assume only 13-1 values. Therefore 2(j3-1),8t-l different numbers can be represented

(the 2 accounts for the positive and negative sign). On the other hand, the exponent can assume U - L + 1 values. Thus, the set JF(j3, t, L, U) contains 2(,8 - l)j3t-l (U - L + 1) different elements.

Solution 1.3 Use the instruction U=2*eye (10) -3*diag(ones (8,1) ,2) (respec- tively, L=2*eye(10)-3*diag(ones(8,1) ,-2)).

Solution 1.4 We can interchange the third and seventh rows of the previous matrix using the instructions: r= [1 : 10]; r (3) =7; r (7) =3; L (r, :). Notice that the character : in L (r , :) ensures that all columns of L are spanned in the L (r , :) usual increasing order (from the first to the last). To interchange the fourth column with the eighth column we can write c=[1:10]; c(8)=4; c(4)=8; L (: , c). Similar instructions can be used for the upper triangular matrix.

Solution 1.5 We can define the matrix A = [v1;v2;v3;v4] where v1, v2, v3 and v4 are the 4 given row vectors. They are linearly independent iff the determinant of A is different from 0, which is not true in our case.

Solution 1.6 The two given functions f and 9 have the symbolic expression:

» syms x » f==sqrt(x A 2+1); pretty(f)

syms pretty

(x 2 +1)1/2

> > g==sin(x"'3)+cosh(x); pretty(g) sin(x 3) + cosh(x) The command syms x allows one to use x as a symbolic variable. The com- mand pretty (f) prints the symbolic expression f in a format that resembles type-set mathematics. At this stage, the symbolic expression of the first and second derivatives and the integral of f can be obtained with the following instructions:

» diff(f,x) ans == 1/(x A 2+1)"'(1/2)*x > > diff(f,x,2) ans == -1/ (x"'2+ 1) '" (3 /2)*x"'2+ 1/ (x"'2+ 1) '" (1/2) » int(f,x) ans == 1/2*x*(x"'2+ 1) '" (1/2)+ 1/2*asi nh (x)

Similar instructions can be used for the function g.

Solution 1.7 The accuracy of the computed roots downgrades as the polyno- mial degree increases. This experiment reveals that the accurate computation of the roots of a polynomial of high degree can be troublesome.

Solution 1.8 Here is a possible program to compute the sequence: function I==sequence(n) 1 == zeros(n+2,1); 1(1) == (exp(l)-l)/exp(l); for i == 0: n, 1 (i + 2) == 1 - (i + 1) * 1 (i + 1); end

The sequence computed from this program doesn't tend to zero (as n in- creases), but it diverges with alternating sign.

Solution 1.9 The anomalous behavior of the computed sequence is due to the propagation of roundoff errors from the innermost operation. In particular, when 4 1 - n z; is less than EM, the elements of the sequence are equal to O. 1'his happens from n == 27.

Solution 1.10 The proposed method is a special instance of a Monte Carlo method and is implemented by the following program: function mypi==pimontecarlo(n) x == rand(n,I); y == rand(n,I); z == x. "'2+y. "'2; v == (z <== 1); m==sum(v); mypi==4*m/n;

The command rand generates a sequence of pseudo-random numbers. The instruction v = (z <= 1) is a shortand version of the following procedure: we check whether z (k) <= 1 for any component of the vector z. If the inequality is satisfied for the k- th component of z (that is, the point (x (k) , y (k)) belongs to the interior of the unit circle) v(k) is set equal to 1, and to 0 otherwise. The command sum (v) computes the sum of all components of v, that is, the sum number of points falling in the interior of the unit circle. By launching the program as mypi=pimontecarlo (n) for different values of n, when n increases, the approximation mypi of 7r becomes more accurate. For instance, for n=1000 we obtain mypi=3 .1120, whilst for n=300000 we have mypi=3.1406.

Solution 1.11 The binomial coefficient can be computed by the following program (see also the MATLAB function nchoosek): nchoosek function bc==bincoeff(n,k) k == fix(k); n == fix(n) ; if k > n, disp('k must be between 0 and n'); break; end if k > n/2, k == n-k; end if k < == 1, b c == n A k; else num == (n-k+l):n; den == l:k; el == num./den; bc == prod(el); end

The command fix (k) rounds k to the nearest integer smaller than k. The command disp (string) displays the string, without printing its name. In general, the command break terminates the execution of for and while loops. If break is executed in an if, it terminates the statement at that point. Finally, prod(el) computes the product of all elements of the vector el.

fix disp break

prod

Solution 2.1 The command fplot allows us to study the graph of the given function j for various values of

. For

== 1, the corresponding function does not have real zeros. For

== 2, there is only one zero, a == 0, with multiplicity equal to 4 (that is, j(a) == j'(a) == j"(a) == j"'(a) == 0, while j(4)(a) -1= 0). Finally, for, == 3, j has two distinct zeros, one in the interval (-3, -1) and the other one in (1,3). In the case, == 2, the bisection method cannot be used

since it is impossible to find an interval (a, b) in which j(a)j(b) < O. For

== 3, starting from the interval [a, b] == [-3, -1], the bisection method (Program 1) converges in 34 iterations to the value a == -1.85792082914850 (with j(a)

-3.6 .10- 12), using the following instructions: > > f==inline('cosh(x)+cos(x)-3'); a==-3; b==-l; tol==l.e-lO; nmax==200; > > [zero, res, niter] == bisection (f, a, b, tol, n max) zero == -1.8579

res == -3.6872e-12 niter == 34

Similarly, for 'Y == 4 the bisection method converges after 34 iterations to the value a == -2.20562163688010, with j(a)

-1.8 .10- 10 .

Solution 2.2 We have to compute the zeros of the function j(V) == pV + aN 2 jV - abN 3 jV 2 - pNb - kNT. Plotting the graph of j, we see that this function has just a simple zero in the interval (0.01,0.06) with j(O.Ol) < 0 and j(0.06) > o. We can compute this zero using the bisection method as follows: > > f==inline('35000000*V+401000jV-17122. 7 jV A 2-1494500', 'V'); > > [zero, res, niter]==bisection(f,O.Ol ,0.06, 1.e-12, 100) zero == 0.0427

res == -6.3814e-05 niter == 35

Solution 2.3 The unknown value of w is the zero of the function j(w) == s(l, w) - 1 == 9.8j2(sinh(w) - sin(w)]jw 2 - 1. From the graph of j we conclude that j has a unique real zero in the interval (0.5,1). Starting from this interval, the bisection method computes the value w == 0.61214447021484 with the desired tolerance in 15 iterations as follows:

> > f==inline('9.8j2*(sinh (omega)- sin(omega)).j omega. A2 -1' , 'omega'); > > [zero, res, niter]==bisection (f,0.5, 1, 1.e-05, 100) zero == 6.1214e-01

res == 3.1051e-06 niter == 15

The Newton method with initial value 0.5 would require only 3 iterations to provide the same significant digits, however with a much smaller residual:

> > fx==inline(' (49 j10*cosh(omega)-49 j10*cos(omega)).j omega. A2_2*... (49 j10*sinh(omega)-49 j10*sin(omega)).j omega. A3' , 'omega') > > [zero, res,niter]==newton(f, fx,0.5, 1.e-05, 100) zero == 6.1214e-01

res == 4.440ge-16 niter == 3

Solution 2.4 The inequality (2.3) can be derived by observing that leCk) I < II C k)I/2 with IICk)1 <

IICk-l)1 < 2-k(b - a). Consequently, the error at the iteration kmin is less than € if kmin is such that 2-krnin-l(b - a) < €, that is, 2-krnin-l < €/(b - a), which proves (2.3).

Solution 2.5 The first formula is less sensitive to the roundoff error.

Solution 2.6 In Solution 2.1 we have analyzed the zeros of the given function with respect to different values of

. Let us consider the case when

== 2. Starting from the initial guess xCO) == 1, the Newton method (Program 2) converges to the value Q == 1.0113e - 04 in 31 iterations with tol=1. e-10 while the exact zero of I is equal to O. This big discrepancy is due to the fact that I is almost a constant in a neighborhood of its zero. Actually, the corresponding residual computed by MATLAB is O. Let us set now

== 3. The Newton method with tol=1. e-16 converges to the value 1.85792082915020 in 10 iterations, while for

== 4 it converges to 2.20562163692958 in 12 iterations (in both cases the residuals are zero in MATLAB).

Solution 2.7 The square and the cube roots of a number a are the solutions of the equations x 2 == a and x 3 == a, respectively. Thus, the corresponding algorithms are: for a given xCO) compute

X Ck + 1) =

(x Ck) + a) k > 0 for the S q uare root , 2 x Ck) , - X Ck + 1) =

(2X Ck) + a) k > 0 for the cube root. 3 (X Ck))2 ' -

Solution 2.8 Setting 8x Ck) == x Ck) - a, from the Taylor expansion of I we find: o = f(a) = f(x Ck ») + 8x Ck) l' (x Ck ») +

(8xCk»)2 f" (x Ck ») + O((8X Ck »)3). (9.1) 2

The Newton method yields

8X Ck + 1) == 8x Ck) - I(x Ck))/ I' (x Ck)).

(9.2)

Combining (9.1) with (9.2), we have 8 Ck+I) =

(8 Ck)) 2 f" (x Ck ») + O((8 Ck»)3) x 2 x f'(x Ck ») x ·

After division by (8X Ck))2 and letting k

00 we prove the convergence result.

Solution 2.9 For certain values of {3 the equation (2.2) can have two roots that correspond to different configurations of the rods system. The two initial values that are suggested have been chosen conveniently to allow the Newton

method to converge toward one or the other root, respectively. We solve the problem corresponding to the following values of (3: (3 == 2k7r 1300 with k == 0, . . . ,100. We use the following instructions to obtain the solution of the problem (shown in Figure 9.1): » a == [10 13 8 10]; > > f == inline(' a(1) 1 a (2)*cos(beta)-a(1) 1 a (4)*cos(alpha)-cos(beta-alpha)+... (a (1) A 2 + a (2) A 2 - a (3) A 2 + a (4) A 2) 1 (2 * a (2) * a (4)) , , , a l ph a ' , , beta' , , a ') ; > > df == inline('a(l)/a(2)*sin(alpha)-sin(beta-alpha)', 'alpha', 'beta', 'a'); > > tol == 1.e-05; xO == -0.1; xl == 2*pi/3; nmax == 200; > > alpha 1 == []; alpha2 == []; niter1 == []; niter2 == []; > > vbeta == linspace(O,2*pi/3,100); > > for beta == vbeta [zero, res, n iter]==newton(f,df,xO, tol, nmax, beta,a); alpha1 == [alpha1,zero]; niter1 == [niter1,niter]; [zero, res, niter] ==newton (f,df,x1, tol, n max, beta,a); alpha2 == [alpha2,zero]; niter2 == [niter2,niter]; end

The components of the vectors alpha1 and alpha2 are the angles computed for different values of (3, while the components of the vectors niter1 and niter2 are the number of Newton iterations (5 or 6) necessary to compute the zeros with the requested tolerance.

- - - - 1.5 .,.....,.,...,...,..;,.....,.,.....,.,... .

...,.,.....,.,.,.,,. i.....,.,.. - - - - -

1 .."...........,,,. -

".."."..,.."..,-,. ....,...-....."".,.-.,.,.....""" - -

, , , . , , , . ,

, , ,

-.;,.-- --... -,:...---- o .,.,.,.,...

'

.

.

+;':'.' .....,

,

.-:. t,

, .

.

.

, .

.

,

....

.,. .....-

-0.5 o

0.5

1.5

2

Fig. 9.1. The two curves represent the two possible configurations which correspond to every value (3 E [0, 27r 13]

Solution 2.10 From an inspection of its graph we see that f has two positive real zeros (a2

1.5 and a3

2.5) and one negative (a1 rv -0.5). The Newton method converges in 4 iterations (having set x(O) == -0.5 and tol = 1. e-10) to the value a1:

> > f==inline('exp(x)-2*x A 2'); df==inline('exp(x)-4*x'); xO==-0.5; tol==1.e-10; > > nmax==100; format long; [zero,res,niter]==newton(f,df,xO,tol,nmax) zero == -0.53983527690282

res ==

9.2 Chapter 2 213

o niter == 4

The given function has a maximum at x

0.3574 (which can be obtained by applying the Newton method to the function f'): for x(O) < x the method converges to the negative zero. If x(O) == x the Newton method cannot be applied since f' (x) == O.

Solution 2.11 Let us set x(O) == 0 and tol== 10- 17 . The Newton method converges in 32 iterations to the value 0.64118239763649, which we identify with the exact zero a. We can observe that the (approximate) errors x(k) - a, for k == 1,2,. . . ,31, decrease only linearly when k increases. This behavior is due to the fact that a has

multiplicity greater than 1 (see Figure 9.2). To recover a second-order method we can use the modified Newton method.

10° 10- 1 10- 2 10- 3 10-4 10- 5 10-6 10 25 30 0 5 15 20

Fig. 9.2. Error vs iteration number of the Newton method for the computation of the zero of the function f(x) == x 3 - 3x 2 2- x + 3x4- x - 8- x

S olution 2.12 We should compute the zero of the function f(x) == sin(x) - V 2gh/v6. From an inspection of its graph, we can conclude that f has one zero in the interval (0,7r/2). The Newton method with x(O) == 7r/4 and tol== 10- 10 converges in 5 iterations to the value 0.45862863227859.

Solution 2.13 Using the data given in the exercise, the solution can be ob- tained with the following instructions:

» f==inline('M-v*(l+l).*((l+l).A n - l)./l','l','M','v','n'); > > df== i n li ne(' - v* ((1 + l). A n-1). / 1- n *v* (1 + l). An. / l +v* (1 + l) . * ((1 + l). A n-1). / l. A 2' , .. . ' 1 ' ' M ' , " ') 0 , , v ,n ,

> > [zero, res, niter]==bisection (f,O.Ol ,0.1, 1.e-12,4,6000, 1000,5);
> > [zero, res,niter]==newton(f,df,zero, 1.e-12, 100,6000, 1000,5);
The Newton method converges to the desired result in 3 iterations.

Solution 2.14 By a graphical study, we see that (2.15) is satisfied for a value of a in (7r/6,7r/4). The Newton method provides the approximate value

0.59627992746547 in 5 iterations, starting from x(O) == 7r 14. We deduce that the maximum length of a rod that can pass in the corridor is L == 30.84.

Solution 2.15 If a is a zero of f with multiplicity m, then there exists a function h such that h(a) -=1= 0 and f(x) == h(x)(x - a)m. The desired result can be obtained by writing both f and its first derivative in the Newton method in terms of h. We obtain for the iteration function the following expression

<J>N X = X _ h(x)(x - a)m ( ) h'(x)(x - a)m + mh(x)(x - a)m

I

lL(x)(x - a) ==x- h'(x)(x - a) + mh(x)

By computing its first derivative, we have:

,1-,.' ( ) = 1 _ [h'(x)(x - a) + h(x)]D(x) - h(x)(x - a)D'(x) 'f'N x [D(x)]2 , where D(x) == h'(x)(x - a) +mh(x). Therefore fjJ

(a) == I-11m and thus the Newton method converges quadratically only if m == 1.

Solution 2.16 Let us inspect the graph of f by using the following com- mands:

» f== 'x A 3+4*x A 2-10'; fplot(f,[-lO,lO]); grid on; » fplot(f,[-5,5]); grid on; » fplot(f,[0,5]); grid on

We can see that f has only one real zero, equal approximately to 1.36 (see Figure 9.3). The iteration function and its derivative are: <J>(x) = 2x 3 + 4x 2 + 10 3x 2 + 8x ' <J>'(x) = (6x 2 + 8x)(3x 2 + 8x) - (6x + 8)(2x 3 + 4x 2 + 10) (3x 2 + 8x) 2 ' and fjJ(a) == a. We easily deduce that fjJ' (a) == 0 by noting that fjJ' (x) == (6x + 8)f(x)/(3x 2 + 8X)2. Consequently, the proposed method converges (at least) quadratically.

Solution 2.17 The proposed method is convergent at least with order 2 since fjJ'(a) == O.

Solution 2.18 By keeping the remaining parameters unchanged, the method converges after only 3 iterations to the value 0.64118573649623 which differs by less than 10- 9 from the result previously computed. However, the behavior of the function, which is quite flat near to x == 0, suggests that the result computed previously could be more accurate. In Figure 9.4 we show the graph

of f in (0.5,0.7), obtained with the following instructions: » f=='x A 3-3*x A 2*2 A (-x) + 3*x*4 A (-x) - 8 A (_x)'; > > fplot(f, [0.5 0.7]); grid on

9.3 Chapter 3 215

0.5

1.5

2

Fig. 9.3. Graph of f(x) == X3 + 4X2 - 10 for x E [0,2]

-3 1 x 10

0.55

0.6

0.65

0.7

Fig. 9.4. Graph of f(x) == x 3 - 3x 2 2- x + 3x4- x - 8- x for x E [0.5,0.7]

Solution 3.1 Since x E (xo, x n), there exists an interval Ii == (Xi-I, Xi) such that x E Ii. We can easily see that maxxE1i I(x - Xi-I)(X - xi)1 == h 2 /4. If we bound Ix - Xi+11 above by 2h, Ix - Xi-21 by 3h and so on, we obtain the inequality (3.6).

Solution 3.2 In all cases we have n == 4 and thus we should estimate the fifth derivative of each function in the given interval. We find: maxxE[-l,l] If?) I < I (5) I I (5) I d 1.76, maxxE[-l,l] f2 < 1.55, max x E[-7r/2,7r/2] f3 < 1.42. The correspon - ing errors are therefore bounded by 0.0028, 0.0024 and 0.0212, respectively.

Solution 3.3 Using the command polyfit we compute the interpolating polynomials of degree 3 in the two cases:

» years==[1975 1980 1985 1990]; > > east==[70.2 70.2 70.3 71.2]; » west==[72.8 74.2 75.2 76.4]; > > ceast==polyfit(years,eor,3); > > cwest==polyfit(years,eoc,3); > > esteast==polyval(ceast,[1970 1983 1988 1995]) esteast == 69.6000 70.2032 70.6992 73.6000

> > estwest==polyval(cwest,[1970 1983 1988 1995]) estwest == 70.4000 74.8096 75.8576 78.4000

Thus, for Western Europe the life expectation in the year 1970 is equal to 70.4 years (estwest(1)), with a discrepancy of 1.4 years from the real value. The symmetry of the graph of the interpolating polynomial suggests that the estimation for the life expectation of 78.4 years for the year 1995, can be overestimated by the same quantity (in fact, the real life expectation is equal to 77.5 years). A different conclusion holds concerning Eastern Europe. Indeed, in that case the estimation for 1970 coincides exactly with the real value, while the estimation for 1995 is largely overestimated (73.6 years instead of 71.2).

Solution 3.4 We chose the month as time-unit. The initial time to == 1 corre- sponds to November 1987, while t7 == 157 to November 2000. With the follow- ing instructions we compute the coefficients of the polynomial interpolating the given prices:

> > time == [1 14 37 63 87 99 109 157]; > > price == [4.5 5 6 6.5 7 7.5 8 8]; » [c] == polyfit(time,price,7); Setting [price2002] = polyval (c, 180) we find that the estimated price of the magazine in November 2002 is approximately 10.8 euros.

Solution 3.5 The interpolatory cubic spline, computed by the command spline in this special case, coincides with the interpolating polynomial. This wouldn't be tFue for the natural interpolating cubic spline.

Solution 3.6 We use the following instructions:

> > T == [4:4:20]; > > rho==[1000.
7794,1000.6427,1000.2805,999.7165,998.9700]; » Tnew ==
[6:4:18]; format long e; > > rhonew == spline(T,rho,Tnew) rhonew ==
Columns 1 through 3 1.000740787500000e+03
1.000488237500000e+03 1.000022450000000e+03 Column 4
9.993649250000000e+02

The comparison with the further measures shows that the approximation is extremely accurate. Note that the state equation for the sea-water (UNESCO, 1980) assumes a fourth-order dependence of the density on the temperature. However, the coefficient of the fourth power of T is of order of 10- 9 .

Solution 3.7 To compute the interpolatory cubic spline we use the following instructions:

4 6 x 10

:

.

_i_

:t;

: o ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,':',,,,,,,,,,,,":' """""""",',"""""""",'' """""""""""""""""".. . : : : . : -2 "",,' r" 1"" "',',,',' j' """""" "" ,

"""" ,,,,··, '\"" "",'" ,

"""""""" "'

"""""" """ ': -: - . : - : -4 "" ":"""

""""" """"" i"",,,,,,,,,,

,,,,,,,,,,,, ,

"""" " ,'" ,

"" """""""

" """,,', . : I I : -6 ..', J"""

".." .,.,...,..,... ",.....'.. ",."..

,."....,.,. ':',...,..,....:,..,..."... . : I : I - 1

1965 1970 1975 1980 1985 1990 1995

Fig. 9.5. The cubic spline (continuous line) and interpolating polynomial (dotted line) for the data of Exercise 3.7. The circles denote the values used in the interpolation

> > year==[1965 1970 1980 1985 1990 1991]; > > production== [17769 24001 25961 34336 29036 33417]; > > years==[1962 1977 1992]; > > estimateprod==spline(year, production ,years); We obtain the following values: 5146.1 xl0 5 Kg in 1962; 22641.7 xl0 5 Kg in 1977; 41894.4 x 10 5 Kg in 1992. The comparison with the real data (12380, 27403 and 32059 x 10 5 Kg, respectively) shows that the values predicted by the spline are inaccurate outside the interpolation interval (see Figure 9.5). Nonetheless, the control exerted by the spline at the end-points of the inter- polation interval allows us to obtain reasonable values also for 1965. On the contrary, the interpolating polynomial introduces large oscillations near this end-point and underestimates the production of as many as -77685 x 10 5 Kg for 1962.

Solution 3.8 The interpolating polynomial p and the spline 83 at 21 eq- uispaced nodes in [-1,1], can be evaluated in 201 equispaced nodes by the following instructions:

> > pert == 1.e-04; > > x== [-1 : 2/20: 1]; y== si n (2 * pi *x) + (-1) . A [1 : 21] * pe rt ; z== [-1 : 0 . 0 1 : 1] ; > > c==polyfit(x,y,20); p==polyval(c,z); s3==spline(x,y,z); When we use the unperturbed data (pert=O) the graphs of both p and 83 are indistinguishable from that of the given function. The situatioIl changes dramatically when the perturbed data are used (pert=!. e-04). In particular, the interpolating polynomial shows strong oscillations at the end-points of the interval, whereas the spline remains practically unchanged (see Figure 9.6). This example shows that approximation by splines is in general more stable with respect to perturbation errors.

Solution 3.9 If n == m, setting! == IInf we find that the first member of (3.18) is null. Thus in this case IInf is the unique solution of the least-square problem.

-0.5

o

0.5

Fig. 9.6. The interpolating polynomial (dotted line) and the interpolatory cubic spline (continuous line) corresponding to the perturbed data. Note the severe oscillations of the interpolating polynomial near the end-points of the interval

Solution 3.10 The coefficients (obtained by the command polyfit) of the requested polynomials are (only the first 4 significant digits are shown): K == 0.67, a4 == 6.301 10- 8 , a3 -8.320 10- 8 , a2 -2.850 10- 4 , al 9.718 10- 4 , ao == -3.032; K == 1.5, a4 == -4.225 10- 8 , a3 == -2.066 10- 6 , a2 == 3.444 10- 4 , al 3.36410- 3 , ao == 3.364; K == 2, a4 == -1.012 10- 7 , a3 -1.431 10- 7 , a2 6.988 10- 4 , al -1.060 10- 4 , ao == 4.927; K == 3, a4 == -2.323 10- 7 , a3 == 7.980 10- 7 , a2 1.420 10- 3 , al == -2.605 10- 3 , ao == 7.315. In Figure 9.7 we show the graph of the polynomial computed using the data in the first column of Table 3.1.

-3

-3.05 .. ...., ,.... ,...,..... ..

,.." "..

-3.1 .....,...i,.,..,.....,.i,."

-3.15

-3.2".....,,

..

-3.35

-3.4 -60 -40 -20 0 20 40 60 80

Fig. 9.1. Least-square polynomial of degree 4 (continuous line) compared with the data in the first column of Table 3.1

Solution 3.11 By repeating the first 3 instructions reported 'in Solution 3.7 and using the command polyfit, we find the following values (in 10 5 Kg): 15280.12 in 1962; 27407.10 in 1977; 32019.01 in 1992,

which represent good approximations to the real ones (12380, 27403 and 32059, respectively).

9.4 Chapter 4 219

Solution 3.12 We can rewrite the coefficients of the system (3.20) in terms of mean and variance by noting that the variance can be expressed as v == I

n 2 M 2 n+l L..."i==O Xi - ·

Solution 3.13 The desired property is deduced from the first equation of the system that provides the coefficients of the least-squares straight line.

Solution 4.1 Using the following third-order Taylor expansions of f at the point Xo, we obtain

4f(xI) = 4f(xo) + 4hf' (xo) + 2h 2 f" (xo) +

h 3 1"' (

), - f(X2) = - f(xo) - 2hf' (xo) - 2h 2 f" (xo) - 4 h 3 fIll C'1). 3 with

E (XO, XI) and rJ E (xo, X2) as two suitable points. Summing this two expressions yields

-3f(xo) + 4f(Xl) - f(X2) = 2hf'(xo) + 2

3 (1'''(0 - 2f'''(1])).

Dividing by 2h and using the mean value theorem, we can conclude the exercise with

o E (xo, X2). A similar procedure can be used for the formula at Xn.

Solution 4.2 Taylor expansions yield

h 2 h 3 f(x + h) = f(x) + hl' (x) + 2 f" (x) + (5 flll (

), f (x - h) = f (x) - hl' (x) +

2 f" (x) -

3 flll (1]) ,

where

and rJ are suitable points. Subtracting these two expressions and dividing by 2h we obtain the result (4.9).

Solution 4.3 Assuming that f E 0 4 and proceeding as in Solution 4.2 we obtain the following errors (for suitable points

l,

2 and

3):

a. - : /4)(6)h 3 , b. - 1

/4)(6)h3,

c. J:... /4)(6)h 4 . 30

Solution 4.4 Using the approximation (4.8), we obtain the following values:

t (months) 0 0.5 1 1.5 2 2.5 3 bn -- 78.04 44.95 19.28 7.02 2.39 -- n' -- 77. 78 39.09 15.19 5.29 1.77 --

By comparison with the exact values of n' (t) we can conclude that the com- puted values are sufficiently accurate.

Solution 4.5 The quadrature error can be bounded by

(b - a)3/(24M 2) max If"(x)l, xE [a,b]

where [a, b] is the integration interval and M the (unknown) number of subin- tervals. The fUIlction fl is infinitely differentiable. From the graph of f{' we infer that I f{' (x) I < 2 in the integration interval. Thus the integration error for fl is less than 10- 4 provided that 5 3 /(241\1 2)2 < 10- 4 , that is M > 322. Also the function f2 is differentiable to any order. Since max x E[O,7r] If

'(x)1 == V2e 3 / 47r , the integration error is less than 10- 4 provided that M > 439. These inequalities actually provide an over estimation of the integration errors. In- deed, the (effective) minimum number of intervals which ensures that the error is below the fixed tolerance of 10- 4 is much lower than that predicted by our result (for instance, for the function fl this number is 51). Finally, we note that since f3 is not differentiable in the integration interval, our theoretical error estimate doesn't hold.

Solution 4.6 On each interval 1 k , k == 1,..., M, the error committed is equal to H 3 /24f" (

k) with

k E (Xk-l, Xk) and hence the global error will be H 3 /24

1 f" (

k). Since f" is a continuous function in (a, b) there exists a point

E (a, b) such that f" (

) ==

1 f" (

k) . Using this result and the fact that M H == b - a, we derive equation (4.13).

Solution 4.7 This effect is due to the accumulation of local errors on each sub-interval.

Solution 4.8 By construction, the mid-point formula integrates exactly the constants. To verify that the linear polynomials also are exactly integrated, it is sufficient to verify that 1 (x) == 1 PM (x). As a matter of fact we have

b J b2 - a 2 J (x) = x dx = 2 '

b+a JpM(X) = (b - a) 2 ·

a

Solution 4.9 For the function fl we find M == 71 if we use the trapezoidal formula and only M == 7 for the Gauss formula. Indeed, the computational advantage of this latter formula is evident.

Solution 4.10 Equation (4.16) states that the quadrature error for the com- posite trapezoidal formula with H = HI is equal to C H'f, with C = - b ;2 a j" (

). If f" does not vary "too much" , we can assume that also the error' with H == H 2 behaves like C H

. Then, by equating the two expressions

I(f)

II + CHr, I(f)

1 2 + CH

,

(9.3)

we obtain C == (II - I 2)/(Hi - Hf). Using this value in one of the expres- sions (9.3), we obtain equation (4.29), that is, a better approximation than is produced by II or 1 2 .

Solution 4.11 We seek the maximum positive integer p such that IappTox(xP) == I(x P). For p == 0,1,2,3 we find the following non- linear system with 4 equations in the 4 unknowns 0:, {3, x and z:

p==O ---+ p==1 ---+

0: + {3 == b - a, b 2 _ a 2 ax + (3 z = 2 ' b 3 - a 3 o:x 2 + {3 z2 == 3 ' b 4 _ a 4 ax 3 + (3z3 = 4 ·

p==2 ---+

p==3 ---+

From the first two equations we can eliminate 0: and z and reduce the system to a new one in the unknowns {3 and x. In particular, we find a second-order equation in {3 from which we can compute (3 as a function of x. Finally, the non-linear equation in x can be solved by the Newton method, yielding two values of x that are the abscissae of the Gauss quadrature points.

Solution 4.12 Since

(4) 24 11 (x) = (1 + (x - 7I")2)5(2x - 271")4 24 + (1 + (x - 71")2)3' fJ4) (x) == -4eXcos(x),

72 (1 + (x - 7r)2)4(2x - 27r)2

we find that the maximum of lfi 4)(x)1 is bounded by M 1

25, while that of 11J4) (x) I by M2

93. Consequently, from (4.22) we obtain H < 0.21 in the first case and H < 0.16 in the second case.

Solution 4.13 Using the command int('exp(-x-2/2)' ,0,2) we obtain for the integral at hand the value 1.19628801332261. The Gauss formula (4.20) applied to the same interval would provide the value 1.20278027622354 (with an absolute error equal to 6.4923e-03), while the Simpson formula gives 1.18715264069572 with a slightly larger error (equal to 9.1354e-03) .

Solution 4.14 We note that Ik > 0 Vk, since the integrand is non-negative. Therefore, we expect that all the values produced by the recursive formula should be non-negative. Unfortunately, the recursive formula is unstable to the propagation of roundoff errors and produces negative elements:

> > 1 (1) == 1 I exp (1); for k == 2: 20 , 1 (k) == 1- k * 1 (k -1); end > > 1(20) -30.1924

Using the composite Simpson formula, with H < 0.25, we can compute the integral with the desired accuracy.

Solution 4.15 For the Simpson formula we obtain

II == 1.19616568040561, 1 2 == 1.19628173356793, =} IR == 1.19628947044542,

with an absolute error in IR equal to -1.4571e-06 (we earn two orders of mag- nitude with respect to II and a factor 114 with respect to 12). Using the Gauss formula we obtain (the errors are reported between brackets):

II == 1.19637085545393 1 2 == 1.19629221796844 IR == 1.19628697546941

(-8.2842e - 05), (-4.2046e - 06), (1.037ge - 06).

The advantage of using the Richardson extrapolation method is evident.

Solution 4.16 We must compute by the Simpson formula the values j(r) == a l(cor 2) J; f(

)d

with r == kilO, for k == 1, . . . ,10 and f(

) == e

2. In order to estimate the integration error we need the fourth derivative f(4)(

) == e

(

2 + 8

+ 12). The maximum of f(4) in the integration interval (0, r) is attained at

== r since f(4) is monotonically increasing. Then we obtain the following values:

> > r== [0.1 : 0.1 : 1] ; > > maxf4==exp(r). *(r. "2+8*r+ 12); maxf4 == Columns 1 through 7 14.1572 16.6599 19.5595 22.9144 26.7917 31.2676 36.4288 Columns 8 through 10 42.3743 49.2167 57.0839 For a given r the error is below 10- 10 provided that H; < 10- 10 2880/(r f(4) (r)). For r == kilO with k == 1, . . . , 10 by the following instructions we can compute the minimum numbers of subintervals which ensure that the previous inequal- ities are satisfied. The components of the vector M contain these numbers:

» x==[0.1:0.1:1]; f4==exp(x).*(x. A 2+8*x+12); > > H==(10 A (-10)*2880. I (x. *f4)). A (1/4); M==fix(x./H) M== 4 11 20 30 41 53 67 83 100 118

Therefore, the values of j (r) are: > > sigma==0.36; epsilonO == 8.85ge-12; f == in Ii n e (, exp (x) . *x. A 2') ;

9.5 Chapter 5 223

for k == 1:10 r == kilO; j(k)==simpsonc(0, r, M (k) ,f); j(k) == j(k)*sigma/r*epsilonO; end

Solution 4.17 We compute E(213) using the Simpson composite formula by increasing the number of intervals until the difference between two consecutive approximations (divided by the last computed value) is less than 10- 11 : > > f==inline('2.3ge-11. I ((x. A 5). *(exp(1.432. I (T*x))-1))', 'x', 'T'); > > a==3.e-04; b==14.e-04; T ==213; > > i == 1 ; err == 1; 10 I d == 0; wh i Ie err > == 1. e-11 I==simpsonc(a, b, i ,f, T); err == abs(I-lold)/abs(I); lold==I; i==i+1; end

The procedure returns the value i == 59. Therefore, using 58 equispaced in- tervals we can compute the integral E(213) with ten exact significant digits. The same result could be obtained by the Gauss formula using 53 intervals. Note that as many as 1609 intervals would be nedeed if using the composite trapezoidal formula.

Solution 4.18 On the whole interval the given function is not regular enough to allow the application of the theoretical convergence result (4.22). One pos- sibility is to decompose the integral into the sum of two intervals, (0,0.5) and (0.5,1), in which the function is regular (it is

actually a polynomial of degree 3). In particular, if we use the Simpson rule on each interval we can even integrate f exactly.

Solution 5.1 The number rk of algebraic operations (sums, subtractions and multiplications) required to compute a determinant of a matrix of order k > 2 with the Laplace rule (1.8), satisfies the following difference equation: rk - krk-l == 2k - 1, with rl == o. Multiplying both side of this equation by llk!, we obtain rk rk-l 2k - 1 -- k! (k - l)! k! Summing both sides from 2 to n gives the solution: n 2k _ 1 n-l 2k + 1 Tn = nIL k! = nIL (k + 1)! ' k=2 k=1

n > 1.

Solution 5.2 We use the following MATLAB commands to compute the de- terminants and the corresponding CPU-times:

» t == [ ]; for i == 3:500 A == magic(i); tt == cputime; d==det(A); t==[t, cputime-tt]; end

The coefficients of the cubic least-square polynomial that approximate the data n= [3: 500] and tare » format long; c==polyfit(n,t,3) c== 0.00000002102187 0.00000171915661 -0.00039318949610 0.01055682398911

The first coefficient (that multiplies n 3), is small, but not small enough with respect to the second one to be neglected. Indeed, if we compute the fourth degree least-square polynomial we obtain the following coefficients:

> > c==polyfit(i,t,4) c== Columns 1 through 4 -0.00000000000051 0.00000002153039 0.00000155418071 -0.00037453657810 Column 5 0.01006704351509

From this result, we can conclude that the computation of a determinant of a matrix of dimension n requires approximately n 3 ops.

Solution 5.3 We have: detA l == 1, detA 2 == £, detA 3 == detA == 2£ + 12. Consequently, if £ == 0 the second principal submatrix is singular and the Proposition 5.1 cannot be applied. The matrix is singular if £ == -6. In this case the Gauss factorization yields

1 0 0 L== 2 1 0 3 1.25 1

1 7 3 U == 0 -12 -4 o 0 0

Note that U is singular (as we could have predicted since A is singular).

Solution 5.4 At step 1, n - 1 divisions were used to calculate the llk entries for i == 2,..., n. Then (n - 1)2 multiplications and (n - 1)2 additions were used to create the new entries a

), for j = 2, . . . , n. At step 2, the numbers of divisions is (n - 2), while the numbers of multiplications and addictions will be (n-2)2. At final step n-1 only 1 addiction, 1 multiplication and 1 division is required. Thus, using the identies

ts = q(q: 1) , 8=1

2 _ q(q + 1)(2q + 1) L- s - 6 ' q > 1, 8=1

9.5 Chapter 5 225

we can conclude that to complete the Gaussian factorization 2(n - l)n(n + 1)/3 + n(n - 1) OpS are required. Neglecting the lower order terms, we can state that the Gaussian factorization process has a cost of 2n 3 /3 opS.

Solution 5.5 By definition, the inverse X of a matrix A E JRnxn satisfies XA == AX == I. Therefore, for j == 1,..., n the column vector Yj of X is the solution of the linear system AYj == ej, where ej is the j-th vector of the canonical basis of JRn with all components equal to zero except the j-th that is equal to 1. After computing the LU factorization of A, the computation of the inverse of A requires the

solution of n linear systems with the same matrix and different right-hand sides.

Solution 5.6 Using the Program 7 we compute the following factors:

L==

100 210 3 -3.38 . 10 15 1

U==

1 1 o -8.88 · 10- 16 o 0

3 14 4.73 · 10- 16

If we compute their product we obtain the matrix

> > L *U ans == 1. 0000 2. 0000 3. 0000

1.0000 3.0000 2. 0000 20. 0000 6.0000 -2.0000

which differs from A since the entry in position (3,3) is equal to -2 while in A it is equal to 4.

Solution 5.7 Usually, only the triangular (upper or lower) part of a sym- metric matrix is stored. Therefore, any operation that does not respect the symmetry of the matrix is not optimal in view of the memory storage. This is the case when row pivoting is carried out. A possibility is to exchange si- multaneously rows and columns having the same index, limiting therefore the choice of the pivot only to the diagonal elements. More generally, a pivoting strategy involving exchange of rows and columns is called complete pivoting (see, e.g., [QSSOO, Chap. 3]).

Solution 5.8 The Land U factors are: 1 0 0 2 -2 0 L== (e; - 2)/2 1 0 U== 0 e; 0 0 -1/e; 1 0 0 3

When e; ---+ 0 l32 ---+ 00. In spite of that, the solution of the system is accurate also when e; tends to zero as confirmed by the following instructions:

> > e==l; for k==1:10 b== [0; e; 2]; L==[l 0 0; (e-2)*0.5 1 0; 0 -lie 1]; U==[2 -2 0; 0 e 0; 0 0 3]; y==L \b; x==U\y; err(k)==max(abs(x-ones(3,1))); e==e*O.l; end > > err err == o

o

o

o

o

o

o

o

o

o

o

Solution 5.9 The computed solutions become less and less accurate when i increases. Indeed, the error norms are equal to 2.63 · 10- 14 for i == 1, to 9.89 · 10- 10 for i == 2 and to 2.10 · 10- 6 for i == 3. This can be explained by observing that the condition number of Ai

increases as i increases. Indeed, using the command cond we find that the condition number of Ai is

10 3 for i == 1,

10 7 for i == 2 and

1011 for i == 3.

Solution 5.10 If (A, v) are an eigenvalue-eigenvector pair of a matrix A, then A 2 is an eigenvalue of A 2 with the same eigenvector. Indeed, from A v == A v follows A 2v == AA v == A 2V. Consequently, if A is symmetric and positive definite K(A 2) == (K(A))2.

Solution 5.11 The iteration matrix of the Jacobi method is:

BJ ==

o o -1 -a

o o o

_0- 1 o o

Its eigenvalues are {O, 0-1, _a- I }. Thus the method converges if 101 > 1. The iteration matrix of the Gauss-Seidel method is

Bcs ==

o 0 o 0 o 0

_0- 1 o 0- 2

with eigenvalues {O, 0, a- 2 }. Therefore, the method converges if 101 > 1. In particular, since p(Bcs) == [p(BJ)]2, the Gauss-Seidel converges more rapidly than the Jacobi method.

Solution 5.12 A sufficient condition for the convergence of the Jacobi and the Gauss-Seidel methods is that A is strictly diagonally dominant. The second row of A satisfies the condition of diagonal dominance provided that 1,81 < 5. Note that if we require directly that the spectral radii of the iteration matrices are less than 1 (which is a sufficient and necessary condition for convergence), we find the (less restrictive) limitation 1,8\ < 25 for both meth.ods.

Solution 5.13 The relaxation method in vector form is

(I - wD- 1 E)x(k+1) == [(1 - w)1 + wD- 1 F]x(k) + wD- 1 b

where A == D - E - F, D being the diagonal of A, and E and F the lower (resp. upper) part of A. The corresponding iteration matrix is

B(w) == (I - wD- 1 E)-1[(1 - w)I + wD- 1 F].

If we denote by Ai the eigenvalues of B(w), we obtain

n

II Ai = Idet [(1 - w)I + wD-IF] I = 11 - win, i==I

Therefore, at least one eigenvalue must satisfy the inequality IAiI > 11 - wi. Thus, a necessary condition to ensure convergence is that 11 - wi < 1, that is, o < w < 2.

Solution 5.14 The given matrix is symmetric. To verify whether it is also definite positive, that is, ZT Az > 0 for all z =I 0 of JR2, we use the following instructions:

> > syms zl z2 real > > z==[zl;z2]; A==[3 2; 2 6]; > > pos==z'*A*z; simple(pos) ans == 3*zl A2+4*zl *z2+6*z2 A 2

The command syms zl z2 real is necessary to declare that the symbolic variables zl and z2 are real numbers, while the command simple (pas) tries several algebraic simplifications of pas and returns the shortest. It is easy to see that the computed quantity is positive since it can be rewritten as 2* (zl +z2) ....2 +zl....2+4*z2....2. Thus, the given matrix is symmetric and positive definite, and the Gauss-Seidel method is convergent.

Solution 5.15 We find:

for the Jacobi method:

{ (I) Xl (1) X 2

=

(1 - x

O)), { _ _.! ((0») . * - 3 Xl ,

(1) Xl (1) X 2

1 - 4' - .!. 3 '

for the Gauss-Seidel method:

{ (I) Xl (1) X 2

=

(1 - x

O)), { xiI) 1 (1) * (1) == -3 X 1 , X 2

1 - 4' 1 12 ·

For the gradient method, we first compute the initial residual

reO) = b - Ax(O) = [

] _ [

] x(O) = [=

j

] .

Then, since

p-I = [

1/2 o

1

3] ,

228 9. Solutions of the exercises

we have z(O) == p-I r (O) == (-3/4, -5/6)T. Therefore (z(O)) T reO) 77 aD = (Z(D») T Az(D) 107 '

and

X(I) == x(O) + QOz(O) == (197/428, -32/321)T.

Solution 5.16 In the stationary case, p(Ba) == minl1 - QAI, where A are the A eigenvalues of p-I A. The optimal value of Q is obtained

solving the equation 11 - QArnin I == 11 - QA rnax I, that is 1 - QArnin == -1 + QArnax, which yields (5.39). Since,

p(Ba) == 1 - QArnin V Q < Qopt,

for Q == Qopt we obtain (5.44).

Solution 5.17 In this case the matrix associated to the Leontieff model is not positive definite. Indeed, using the following instructions:

» for i==1:20; for j==1:20; c(i,j)==i+j; end; end; A==eye(20)-c; » min(eig(A)) ans == -448.5830 » max(eig(A)) ans == 30.5830

we can see that the minimum eigenvalue is a negative number and the maxi- mum eigenvalue is a positive number. Therefore, the convergence of the gra- dient method is not guaranteed. However, since A is non singular, the given system is equivalent to the system AT Ax == A Tb, where AT A is symmetric and positive definite. We solve the latter by the gradient method requiring that the norm of the residual be less than 10- 10 and starting from the initial data x(O) == OT:

» b == [1:20]'; aa==A'*A; b==A'*b; xO == zeros(20,1); > > [x,iter]==richardson(aa ,b,xO, 100, 1.e-10); The method converges in 15 iterations. A drawback of this approach is that the condition

number of the matrix A T A is, in general, larger than the condition number of A.

'. .. .. , .. . II.. III1B.lmlM,.: . ... '. . " . ..., ... -. .....

Solution 6.1 AI: the power method converges in 34 iterations to the value 2.00000000004989. A 2 : starting from the same initial vector, the power method

9.6 Chapter 6 229

requires now 457 iterations to converge to the value 1.99999999990611. The slower convergence rate can be explained by observing that the two largest eigenvalues are very close one another. Finally, for the matrix A3 the method doesn't converge since A3 features two distinct eigenvalues (i and -i) of max- imum modulus.

Solution 6.2 The Leslie matrix associated with the values in the table is

A==

o 0.5 0.8 0.3 0.2 0 0 0 o 0.4 0 0 o 0 0.8 0

Using the power method we find A1

0.5353. The normalized distribution of this population for different age intervals is given by the components of the cor- responding unitary eigenvector, that is, Xl

(0.8477,0.3167,0.2367, 0.3537)T.

Solution 6.3 We rewrite the initial guess as y(O) = (3(O) (alXI + a2 X 2 + ?n

aiXi) , with j3 C O) == l/IIx CO) II. By calculations similar to those carried out in Section 6.1, at the generic step k we find: yCk) ==

k j3Ck) (aIXIeik19 + a2x2e- ik19 + ta i .xi Xi) . i==3

The first two terms don't vanish and, due to the opposite sign of the exponents, the sequence of ti-e yCk) oscillates and cannot converge.

Solution 6.4 From the eigenvalue equation Ax == AX, we deduce A -1 Ax == AA -lX, and therefore A -lX == (l/A)x.

Solution 6.5 The power method applied to the matrix A generates an oscil- lating sequence of approximations of the maximum modulus eigenvalue (see, Figure 9.8). This behavior is due to the fact that this eigenvalue is not unique.

Solution 6.6 To compute the eigenvalue of maximum modulus of A we use Program 9: > > A==wi I ki nson (7); > > xO==ones(7,1); tol==1.e-15; nmax==lOO; > > [lambda ,x, iter]==eigpower(A, tol, nmax,xO); After 35 iterations we obtain lambda=3. 76155718183189. To find the largest negative eigenvalue of A, we can use the power method with shift and, in particular, we can choose a shift equal to the largest positive eigenvalue that we have just computed. We find:

20

40

60

80

Fig. 9.8. The approximations of the maximum modulus eigenvalue of the matrix of Solution 6.5 computed by the power method

> > [la m bda2,x, iter] ==eigpower(A-la m bda *eye(7), tol, n max,xO); > > lambda2+lambda ans == -1.12488541976457

after iter = 33 iterations. These results are satisfactory approximations of the largest (positive and negative) eigenvalues of A.

Solution 6. 7 Since all the coefficients of A are real, eigenvalues occur in con- jugate pairs. Note that in this situation conjugate eigenvalues must belong to the same Gershgorin circle. The matrix A presents 2 column circles isolated from the others (see Figure 9.9 on the left). Each of them must contain only one eigenvalue that must therefore be real. Then A admits at least 2 real eigenvalues. Let us consider now the matrix B that admits only one column isolated circle (see Figure 9.9 on the right). Then, thanks to the previous consideration the corresponding eigenvalue must be real. The remaining eigenvalues can be either all real, or one real and 2 complex.

3 ....

....."..,....,..,.....,.. '.. ..;.....'.. . .

1 .....i.....,.. ....,

. . . . . 2 ..,. ,..:.....,....

.....,..... .:..... . . . . . .

. . -1 .....:....,.....,.....

....

-3 2

4

6

0.5... . ' '.....

: :::... . ...::: :::::::: :::::::: ::::. -1 .... ""!"""" ...., 12 -1.

-5 -<4 -3 -2 -1

. . . -2 ..... .

..,.....,..... ..;,.... ..... .

....

10

Fig. 9.9. On the left, column circles of the matrix A of Solution 6.7. On the right, column circles of the matrix B of Solution 6.7

Solution 6.8 The row circles of A feature an isolated circle of center 5 and radius 2 the maximum modulus eigenvalue must belong to. Therefore, we can set the value of the shift equal to 5. The comparison between the number of iterations and the computational cost of the power method with and without shift can be found using the following commands:

> > A==[5 0 1 -1; 0 2 0 -1/2; 0 1 -1 1; -1 -1 0 0]; > > flops(0); [la m bda ,x, iter] ==eigpower(A, tol, n max,xO); flops, iter ans ==

2204

iter == 34 > > flops(O); [lambda,x,iter]==invshift(A,tol,nmax,5,xO); flops, iter ans ==

1082

iter == 12

The power method with shift requires in this case a lower number of it- erations (1 versus 3) and half the cost than the usual power method (also accounting for the extra time needed to compute the Gauss factorization of A off-line) .

Solution 6.9 Using the qr command we have immediately:

> > A==[2 -1/2 0 -1/2; 0 4 0 2; -1/2 0 6 1/2; 0 0 1 9]; > > [Q,R]==qr(A) Q== -0.9701 0.0073 -0.2389 -0.0411 o -0.9995 -0.0299 -0.0051 0.2425 0.0294 -0.9557 -0.1643 o 0 -0.1694 0.9855 R== -2.0616 0.4851 1.4552 0.6063 o -4.0018 0.1764 -1.9881 o 0 -5.9035 -1.9426 o 0 0 8.7981

To verify that RQ is similar to A, we observe that

QT A == QT QR == R

thanks to the orhogonality of Q. Thus C == QT AQ == RQ , since QT == Q-l, and we conclude that C is similar to A.

Solution 6.10 A simple implementation of the proposed algorithm is: function [A]==qrsimple(A,it) for i == 1: it [Q,R]==qr(A); A == R*Q; end

where it represents the number of iterations once the algorithm is stopped. For the given matrix, after 10 iterations we obtain the matrix

A (10):

» A10 == 9.2090 1.1678 1.5237 0.0705 0.1220 4.9228 -0.5739 0.0461 -0.1732 -1.3095 4.8686 -0.6902 0 0 0.0017 1.9996

while, after 50 iterations we obtain the matrix A (50):

» A50 == 9.1728 -0.8113 1.8631 0.0000 5.8003 0.6208 -0.0000 -0.0000 4.0268 o 0 0.0000

0.1059 0.5914 -0.3526 2.0000

Note that the matrices A (k) are similar since A (k+l) == (Q(k))T A (k)Q(k) for k > 0 (see Solution 6.9). Therefore, we can guess that the sequence of matrices A (k) when k

00 tends to an upper triangular matrix whose elements are the eigenvalues of A.

Solution 6.11 We can use the command eig in the following way: [X,D] =eig(A), where X is the matrix whose columns are the unit eigenvectors of A and D is a diagonal matrix whose elements are the

eigenvalues of A. For the matrices A and B of Exercise 6.7 we should execute the following instructions:

> > A==[2 -1/2 0 -1/2; 0 4 0 2; -1/2 0 6 1/2; 0 0 1 9]; > > eig(A) ans == 2.0000 5.8003 9.1728 4.0268 » B==[-5 0 1/2 1/2; 1/2 2 1/2 0; 010 1/2; 0 1/4 1/2 3]; > > eig(B) ans == -4.9921 2.1666 -0.3038 3.1292

... ": ':..... :. . .... :.:..... : ..... . ':-: ...: .:.....: ..:.

:.

. " .' .: :: :. ..:..... -:' ::.; ; ":":"?: ... . '" ., . . . " .. 1 .:..... ... . : 1 ' '.': . 1 ' . ' '":" I ': .

:"" I >'" >" 1 ;:';'> . .." . ".. . "C' 0.. . .....:. . :. ,... .... -m" ,>'.<

:({';"">"

Solution 7.1 Let us approximate the exact solution y(t) ==

[e t - sin(t) - cos(t)] of the Cauchy problem (7.43) by the forward Euler method using dif- ferent values of h: 1/2, 1/4, 1/8,...,1/512. The associated error is computed by the following instructions:

» yO==O; f inline('sin(t)+y','t','y'); y=='O.5*(exp(t)-sin(t)-cos(t))';

> > tspan==[O 1]; N==2; for k==1:10 [tt,u]==feuler(f,tspan ,yO, N) ;t==tt(end) ;e(k)==abs(u(end)-eval(y)); N==2*N ;end » e e== Columns 1 through 7 0.4285 0.2514 0.1379 0.0725 0.0372 0.0189 0.0095 Columns 8 through 10 0.0048 0.0024 0.0012

Now we apply formula (1.11) to estimate the order of convergence: > > p==log(abs(e(1 :end-1). I e(2:end))) Ilog(2) p== Columns 1 through 7 0.7696 0.8662 0.9273 0.9620 0.9806 0.9902 0.9951 Columns 8 through 9 0.9975 0.9988

As expected the order of convergence is one. With the same instructions (sub- stituting the program feuler with the program beuler) we obtain an estimate of the convergence order of the backward Euler method:

> > p==log(abs(e(I :end-1). I e(2:end))) Ilog(2) p== Columns 1 through 7 1.5199 1.1970 1.0881 1.0418 1.0204 1.0101 1.0050 Columns 8 through 9 1.0025 1.0012

Solution 7.2 The numerical solution of the given Cauchy problem by the forward Euler method can be obtained as follows:

> > tspan==[O 1]; N== 100;f==inline(' -t*exp(-y)' ,'t' ,'y') ;yO==O; > > [t, u]==feuler(f,tspan ,yO, N);

To compute the number of exact significant digits we can estimate the con- stants Land M which appear in (7.11). Note that, since f(t,

y(t)) < 0 in the given interval, y(t) == log(1 - t 2 /2) is a monotonically decreasing function, vanishing at t == O. Since f is continuous together with its first derivative, we can approximate L as L == maxo < t < IIL(t)1 with L(t) == af lay == te- Y . Note that L(O) == 0 and L(t) > 0 for all t E (0,1]. Thus, L == e. Similarly, in order to compute M == maxo < t < lly"(t)1 with y" == -e- Y - t 2 e- 2Y , we can observe that this function has its maximum at t == 1, and then M == e + e 2 . From (7.11) we deduce e L - 1 M IUlOO - y(1)1 < L 200 = 0.26. Therefore, there is no guarantee that more than one significant digit be exact. Indeed, we find u(end)=-O. 6785, while the exact solution at t == 1 is y(l) == -0.6931.

Solution 7.3 The iteration function is <p(u) == Un - htn+1e-u and the fixed- point iteration converges if I <P' (u) I < 1. This property is ensured if h(to + (n + l)h) < e U . If we substitute u with the exact solution, we can provide an a priori estimate of the value of h. The most restrictive situation occurs when u == -1 (see Solut ion 7.2). In this case the solution of the inequality (n + 1) h 2 < e -1 is h < J e -1 / (n + 1).

Solution 7.4 We repeat the same set of instructions of Solution 7.1, however now we use the program cranknic (Program 14) instead of feuler. According to the theory, we obtain the following result that shows second-order conver- gence:

> > p==log(a bs(e(1 :end-1). / e(2:end))) /log(2) p== Columns 1 through 7 2.0379 2.0092 2.0023 2.0006 2.0001 2.0000 2.0000 Columns 8 through 9 2.0000 2.0000

Solution 7.5 Consider the integral formulation of the Cauchy problem (7.4) in the interval [tn, t n +1]:

t n +l y(tn+l)-y(t n) J f(T,y(T))dT t n h ::= 2 [f(tn,y(t n)) + f(tn+l,y(tn+t))],

where we have approximated the integral by the trapezoidal formula (4.17). By setting Uo == y(to) and replacing y(t n) by the approximate value Un and the symbol

by ==, we obtain

h Un+l = Un + 2 [f(t n , Un) + f(!n+l, Un+l)] , which is the Crank-Nicolson method.

'in > 0,

Solution 7.6 Set hA == x + iy, so that 11 + h;\1 2 == (1 + X)2 + y2. Then, the assumption 11 + h;\1 < 1 is equivalent to (1 + X)2 + y2 < 1. It follows that the region of absolute stability for the forward Euler method is the unit circle in the complex plane (see Figure 9.11).

Solution 7.7 Thanks to the result found in Solution 7.6, we must impose the limitation 11 - h + ihl < 1, which yields 0 < h < 1.

Solution 7.8 Let us rewrite the Heun method in the following (Runge-Kutta like) form:

1 UnH = Un + 2 (k 1 + k2), k 1 = hf(t n , Un), k2 = hf(tn+l, Un + kt). (9.4)

9.7 Chapter 7 235

We have hTn+l(h) = y(tn+l) - y(t n) - (k 1 + k 2)/2, with k 1 = hf(tn,y(t n)) ..-.. ..-.. and k2 == hf(tn+l, y(t n) + k 1). Therefore, the method is consistent since

lim Tn+l == y' (t n) - 2 1 (f(tn, y(t n)) + f(t n , y(t n))] = O. h-+O

The Heun method is implemented in Program 19. Using this program, we can verify the order of convergence as in Solution 7.1. By the following instruc- tions, we find that the Heun method is second-order with respect to h

> > p==log(abs(e(1 :end-l). 1 e(2:end))) llog(2) p== Columns 1 through 7 1.7642 1.8796 1.9398 1.9700 1.9851 1.9925 1.9963 Columns 8 through 9 1.9981 1.9991

. . function [t, u]==rk2(odefun, tspan ,yO, Nh ,varargin) h==(tspan(2)-tspan(1)-to) lNh; tt==[tspan(I):h :tspan(2)]; u(1)==yO; for s==tt(1 :end-l) t == s; Y == u(end); kl==h*feval(odefun,t,y,varargin{:}); t == t + h; y == y + kl; k2==h*feval(odefun,t,y,varargin{:}); u == [u, u(end) + 0.5*(kl+k2)]; end t==tt. ,

Solution 7.9 Applying the method (9.4) to the model problem (7.19) we obtain k 1 == hAU n and k 2 == hAu n (l + hA). Therefore Un+l == u n [l + hA + (hA)2/2] == u n P2(hA). To ensure absolute stability we must require that lp2(hA)1 < 1, which is equivalent to 0 < p2(hA) < 1, since p2(hA) is positive. Solving the latter inequality, we obtain -2 < hA < 0, that is, h < 2/1Al.

Solution 7.10 Note that

Un == Un-l (1 + hAn-l) + hrn-l.

Then proceed recursively on n.

Solution 7.11 The inequality (7.29) follows from (7.28) by setting 1 1 <p(A) = 1 + A + A ·

The conclusion follows easily.

Solution 7.12 From (7.26) we have

n-l lZn - Un I < prnax an + hprnax L8(h)n-k-l. k==O

The result follows using (7.27).

Solution 7.13 We have

1/'.../'... /'... hTn+l (h) = y(tn+l) - y(t n) - 6 (k 1 + 4k 2 + k 3), /'... /'... h k k 1 == hf(t n , y(t n)), k2 == hf(t n + 2' y(t n) + T)' k3 == hf(tn+l, y(t n) + 2k 2 - k 1).

This method is consistent since

lim Tn+l == y' (t n) - 6 1 [f(tn, y(t n)) + 4f(t n , y(t n)) + f(t n , y(t n))] = O. h-+O

This method is an explicit Runge-Kutta method of order 3 and is imple- mented in Program 20. As in Solution 7.8, we can derive an estimate of its order of convergence by the following instructions:

> > p==log(abs(e(1 :end-1). / e(2:end))) /log(2) p== Columns 1 through 7 2.7306 2.8657 2.9330 2.9666 2.9833 2.9916 2.9958 Columns 8 through 9 2.9979 2.9990

Solution 7.14 From Solution 7.9 we obtain the relation 1 2 1 3 Un+l = Un [1 + hA + 2 (hA) + 6 (hA)] = u n P3(hA).

By inspection of the graph of P3, obtained with the instruction

» c==[1/6 1/2 1 1]; z==[-3:0.01:1]; p==polyval(c,z); plot(z,abs(p)) we deduce that Ip3(h;\)1 < 1 for -2.5 < h;\ < o.

;

)

:

;';:

;

:.

;

.

.

.:.

.

.

:.

function [t, u]==rk3(odefun ,tspan ,yO, N h ,varargin) h==(tspan(2)-tspan(1)) /Nh; tt==[tspan(l):h :tspan(2)]; u(l)==yO; for s==tt(1:end-1) t == s; Y == u(end); t == t + h*0.5; y == y + 0.5*k1;

k1==h*feval(odefun,t,y,varargin{:}); k2==h*feval(odefun,t,y,varargin{:});

9.7 Chapter 7 237

t == 5 + h; y == u(end) + 2*k2-k1; k3==h*feval(odefun,t,y,varargin{:}); u == [u, u(end) + (k1+4*k2+k3)/6]; end t==tt;

Solution 7.15 The method (7.46) applied to the model problem (7.19) gives the equation U n +l == u n (l + h>'" + (h>...)2). From the graph

of 1 + z + Z2 with z == h>'", we deduce that the method is absolutely stable if -1 < h>'" < O.

Solution 7.16 To solve Problem 7.1 with the given values, we repeat the following instructions with N=10 and N=20:

> > f==inline('-1.68*10 A (-9)*y A 4+2.6880', 't', 'y'); > > [t, uc]==cra n kn ic(f, [0,200],180, N); > > [t,u]==predcor(f,[O 200],180,N, 'feonestep', 'cnonestep');

The graphs of the computed solutions are shown in Figure 9.10. The solutions obtained by the Crank-Nicolson method are more accurate than those obtained by the Heun method.

200

..

... ..,....

.

.

..

.

.l.

.

.

.

.j. .-.... .... :, : : . , . -., ... ,,; . . . . :. . . . .; . . . . , : . . ,: : . : . 194 ..... 'I

"""

""""""""" '

""""""""",""""""""

""""""""""?"""""""""

""""""""" :,: .. .: : , : : . . : : 192 ..... .:.,.....;.....

.....;..... i.....;..... i..... i . . : : . : ,: : : . : : : 190 ..... .1.:..... ...:..... .:..... .;..... .:..... ...;..... :..... I : . : : . : : I : :: ::. 188 ....

/":"""""""""

"""""""""

"""""""""

""""""""" j.....i.....

..... I: . : . : : . 186 ... "..... .

.....

..... +.....

.....

.....

.....

..... ,: : : : . : : 184 . I..... . .....:.....

.....:.....

.....

.... " .... . - -- - - . - - 182 ..... .

....

..... i.....:..... .....;..... .?..... .....

..... - . - - . . - . .. - - . . - - . . - . -. --

200

. . . . 194 ....

..... .:.....

.....

.....;..... . - - - - . .. .. . - - . - - . 192 . .. .

.. . .

. .. .

.. ..

. . . .

. ...; - . - . . .. - - - - - . - - -

196 ..

.,...'',...,...,...,..,...

190 ....

..... .:.....

.....:..... ..

.....:... .... - - - - .. . 188 ...
.....;.....:.....:.....:.....:.....:..... - - - . - - - - . - - - - - - . - - -. - - -

186 .. ..

.....:.....

.....

.....

.....

.... -. -. - - - - - . - - - - . .'

182 .....

.....

..... i.....;..... i.....;..... ..... - - . - - . - - - - - . . . . . - . - - . - - - - -

25

50

75 100 125 150 175 200

180 o

25

50

75 100 125 150 175 200

Fig. 9.10. Computed solutions with h == 20 (left) and h == 10 (right) for the Cauchy problem of Solution 7.16: in the continuous line, the solutions computed by the Crank-Nicolson method, in the dashed lines those computed by the Heun method

Solution 7.17 Heun method applied to the model problem (7.19), gives

(1 2 2) U n +l = Un 1 + h)" + 2 h)" ·

In the complex plane the boundary of its region of absolute stability satisfies 11+h>"'+h2 >...2 /212 == 1, having set h>'" == x+iy. This equation is satisfied by the numbers (x,y) such that f(x,y) == X 4 +y4+2x 2 y2+4x 3 +4xy2+8x 2 +8x == o. We can represent this curve as the level curve f(x, y) == z (corresponding to the level z == 0). This can be done by means of the following instructions:

> > f=='x. '" 4+y. "'4+2*(x. "'2). *(y. "'2)+4*x. *y. "'2+4*x. "'3+8*x. "'2+8*x'; > > [x, y] == m es h g rid ([- 2 .1 : 0 .1 : 0.1] , [- 2: 0 .1 : 2]) ; » contour(x,y,eval(f),[O 0]) meshgrid The command meshgrid draws in the rectangle [-2.1,0.1] x [-2,2] a grid with 23 equispaced nodes in the x-direction, and 41 equispaced nodes in the contour y-direction. With the command contour we plot the level curve of f(x, y) (evaluated with the command eval (f)) corresponding to the value z == 0 (made precise in the input vector [0 0] of contour). In Figure 9.11 the continuous line delimitates the region of absolute stability of the Heun method. This region is larger than the corresponding region of the forward Euler method (which corresponds to the interior of the dashed circle). Both curves are tangent to the imaginary axis at the origin (0,0).

2

1.5 . "."

\ , ...\..... ..... .,.. .....

..... , , . , . ,'. : ."..... .

. .... .

.

:

....._

_....".'.

.....,' ,¡".,..,.."..,...., '.. "",... . . -

. . 1 ' , . , . , . . '

' . . , , . , . . . , , ... , . ,

. """":" :.,,; :...a.' -:'-' "".;..'

' , , , , . . : . -, ,- : . , , . . , , 0.5 .,..,." ':"". "". """" ', """""""" ':" "'. " ... "....

". I

o "'."....:." ,'.."..,.., ,.."..,.,....,

-0.5

-1 ..."."."'"

. - -1.5 ....."..:"..,.,...,...,.,.'!. - .

-2

-3

-2

-1

o

Fig. 9.11. Boundaries of the regions of absolute stability for the Heun method (continuous line) and the forward Euler method (dashed line). The correspond- ing regions lie at the interior of the boundaries

Solution 7.18 We use the following instructions:

» tspan==[O 1]; yO==O; f==inline('cos(2*y)','t','y'); > > y== '0.5*asin((exp(4 *t)-1). / (exp(4 *t)+ 1))'; > > N==2; for k==1:10 [tt, u] ==predcor(f, tspa n ,yO, N, 'feonestep' , , cnonestep'); t==tt(end); e(k)==abs(u(end)-eval(y)); N==2*N; end > > p==log(abs(e(1 :end-1). / e(2:end))) /log(2) p== Columns 1 through 7 2.4733 2.2507 2.1223 2.0601 2.0298 2.0148 2.0074 Columns 8 through 9 2.0037 2.0018

As expected, we find that the order of convergence of the method is 2. However, the computational cost is comparable with that of the forward Euler method, which is first-order accurate only.

9.7 Chapter 7 239

Solution 7.19 The second-order differential equation of this exercise is equiv- alent to the following first-order system:

X' == Z Z' == -5z - 6x , ,

with x(O) == 1, z(O) == O. We use the Heun method as follows: » tspan==[O 5]; yO==[l 0]; > > [tt, u]==predcor('fspring', tspan ,yO, N, 'feonestep',' cnonestep'); where N is the number of nodes and

fspring.m is the following function: function y==fspring(t,y) b==5; k==6; yy==y; y(1)==yy(2); y(2)==-b*yy(2)-k*yy(1);

In Figure 9.12 we show the graphs of the two components of the solution, computed with N=20 ,40 and compare them with the graph of the exact solution x(t) == 3e- 2t - 2e- 3t and that of its first derivative.

0.6 .....

. . . . . , . . , . . , . .

. . . . . , , . , ... , , . , . . , . . , , ... , - - - . . . - . - - - . ,.. " , ' .,.....,..... " .....-.....,.. '" .. ,. - . . - - . - . - . - .

0.4 .....

- - ,.....,. - . - - . . - . - . - . - -

, . . . , , ... , . ' \ . , ... , . . , . . , . , . .

. . . . , . . , , , . . , ... , , . . . - - . - - - .

:::

';.:.. ....: .....L>.,

t

,.- ::-l:: :::::::.:: :::: .:!:.: .:: :::::: .:::. ',', ,

l : : : -0.6 "," """,flll'.,

,..

", ,...,..",., ...

...".." "'0"" .

."...,..,.,...., ,.. , G: :: -0.8 . .th;;.;;-'..... ..j..... ....

....

.... - .

-1 o

2

3

4

5

Fig. 9.12. Approximations of x(t) (continuous line) and X'(t) (dashed line) computed with N=20 (thin line) and N=40 (thick line). Small circles and squares refer to the exact functions x (t) and x' (t), respectively

Solution 7.20 The second-order system of differential equations is reduced to the following first-order system:

I X == Z, I Y == V, Z' == 2w sin(w) - k 2 x, v' == -2w sin(W)z - k 2 y.

(9.5)

If we suppose that the pendulum at the initial time to == 0 is at rest in the position (1,0), the system (9.5) must be given the following initial conditions:

x(O) == 1, y(O) == 0, z(O) == 0, v(O) == O.

Setting W == 7r/4, which is the average latitude of the Northern Italy, we use the forward Euler method as follows:

> > [t,y]==feuler('ffocault', [0 300], [1 0 0 0], Nh);

where Nh is the number of steps and ffocault.m is the following function:

```
function y==ffocault(t,y) 1==20; k2==9.8/1; psi==pi/4;
omega==7.29*1.e-05; yy==y; y(l)==yy(3); y(2)==yy( 4);
y(3)==2*omega*sin(psi)*yy( 4 )-k2*yy(l); y( 4
)==-2*omega*sin(psi)*yy(3)-k2*yy(2);
```

By some numerical experiments we conclude that the forward Euler method cannot produce acceptable solutions for this problem even for very small h. For instance, on the left of Figure 9.13 we show the graph, in the phase plane (x, y), of the motion of the pendulum computed with N=30000, that is, h == 1/100. As expected, the rotation plane changes with time, but also the amplitude of the oscillations increases. Similar results can be obtained for smaller h and using the Heun method. In fact, the model problem corresponding to the problem at hand has a coefficient A that is purely imaginary. The corresponding solution (a sinusoid) is bounded for t that tends to infinity, however it doesn't tend to zero. Unfortunately, both the forward Euler and Heun methods feature a region of absolute stability that doesn't include any point of the imaginary axis (with the exception of the origin). Thus, to ensure the absolute stability one should choose the prohibited value h == O. To get an acceptable solution we should use a method whose region of absolute stability includes a portion of the imaginary axis. This is the case, for instance, for the adaptive Runge-Kutta method of order 3, implemented ode23 in the MATLAB function ode23. We can invoke it by the following command:

> > [t,u]==ode23('ffocault' ,[0 300],[1 0 0 0]);

In Figure 9.13 (right) we show the solution obtained using only 1022 integra- tion steps. Note that the numerical solution is in good agreement \\rith the analytical one.

0.04

0.015

-0.02 ..... .:.....

..... .

.....

.... - . - - - - - - . . . . - . -0.03 ..... .

.....

.....

.....

.... - . - . - - . - . - - . - - . .

-0.005 ..... .,. ,. ....., ... " . ...,... ....,... -0.01 ....,.,...,..,....,
.... "..., ...,.. ....,., ,.

....., ,.,.,..,.,., ,.. " :-.....

0.01 ..

. - . ..,.,.....,.....""."..,.....,..."..,.,,._.,0"""""-" . - . .

0.03 ...,.....,.....,..,

. . , ... , ... ,

.. , ...

, ... , ... , . ,? . . , ... , ... - - - . - . . . - - . - - " .

o .....:., ',.....

...r.....,.."..,.."....,.,.,.""....,..,-,...."..... . . . . . -. . .. - - - - . -

0.02 "....."..

""

0.01 ..,.....,...:.....

. . . . , . . , ... , . , , ...

, , , . . . , . . . , ... , ... , . , . . - . - . . - . .

-0.01 .....,.. ':.....'.'.....

,.....,.....

...,.,.,.

-0.04 -3

-2

-1

o

2

-0.015 3 -1

-0.5

o

0.5

Fig. 9.13. Trajectories on the phase plane for the Focault pendulum of So- lution 7.20 computed by the forward Euler method (left) and the third-order adaptive Runge-Kutta method (right)

9.8 Chapter 8 241

Solution 8.1 We can verify directly that x T Ax > 0 for all x =I o. Indeed,

2 -1 0 0 Xl -1 2 X2 [Xl X2 ... XN-I XN] 0 -1 0

-1 2 -1 o 0 -1 2 == 2xi - 2XIX2 + 2x

- 2X2X3 + . . . - 2XN-IXN + 2xJv.

XN-I XN

The last expression is equivalent to (XI-X2)2+.. .+(XN-I-XN)2+xi+x

, which is, positive provided that at least one Xi is non-null.

Solution 8.2 We verify that AQj == AjQj. Computing the matrix-vector prod- uct w == AQj and requiring that w is equal to the vector AjQj, we find:

2sin(jO) - sin(2jO) == 2(1- cos(jO))sin(jO), - sin(jkO) + 2 sin(j(k + 1)0) - sin(j(k + 2)0) == 2(1 - cos(jO)) sin(2jO), k == 1, . . . , N - 2

2sin(NjO) - sin((N - 1)jO) == 2(1 - cos(jO)) sin(NjO). The first equation is an identity since sin(2jO) == 2 sin(jO) cos(jO). The other equations can be simplified since

sin(jkO) == sin((k + 1)jO) cos(jO) - cos((k + 1)jO) sin(jO), sin(j(k + 2)0) == sin((k + 1)jO) cos(jO) + cos((k + 1)jO) sin(jO).

Since A is symmetric and positive definite, its condition number is K(A) == Amax/Amin, that is, K(A) == AI/AN == (1-cos(N1r/(N + 1)))/(1-cos(1r/(N + 1))). Using the Taylor expansion of order 2 of the cosine function, we obtain K(A)

N 2 , that is, K(A)

h- 2 .

Solution 8.3 We note that

h 2 h 3 h 4 u(x + h) = u(x) + hu' (x) + 2 u " (x) + 6 u '" (x) + 24 U(4) (

+), h 2 h 3 h 4 u(x - h) == u(x) - hu ' (x) + -u" (x) - -U'" (x) + _U(4) (

_), 2 6 24 where

+ E (x,x+h) e

- E (x-h,x). Summing the two expression we obtain h 4 u(x + h) + u(x - h) = 2u(x) + h 2 u"(x) + 24 (U(4)(

+) + U(4)(

_)), which is the desired property.

Solution 8.4 The matrix is again tridiagonal with entries ai,i-l == -1 - h%, aii == 2+h21', ai,i+l == -l+h%. The right-hand side, accounting for the bound- ary conditions, becomes f == (f(xl)+a(1+h8/2)/h 2 , f(X2),..., f(XN-l), f(XN)+ 13(1 - h8 /2) / h 2)T.

Solution 8.5 With the following instructions we compute the corresponding solutions to the three given values of h:

> > fbvp==inline(' l+sin(4*pi*x)', 'x'); > > [z, uh 10]==bvp(0,1,9,0,0.1, fbvp,O,O); > > [z,uh20]==bvp(0,1,19,0,0.1,fbvp,0,0); > > [z,uh40]==bvp(0,1,39,0,0.1,fbvp,0,0);

Since we don't know the exact solution, to estimate the convergence order we compute an approximate solution on a very fine grid (for instance h == 1/1000), then we use this latter as a surrogate for the exact solution. We find:

> > [z,uhex]==bvp(O, 1,999,0,0.1 ,fbvp,O,O); > > max(abs(uh9-uhex(1:100:end))) ans == 8.6782e-04 > > max(abs(uh19-uhex(1:50:end))) ans == 2.0422e-04 > > max(abs(uh39-uhex(1:25:end))) ans == 5.278ge-05

Halving h, the error is divided by 4, proving that the convergence order with respect to h is 2.

Solution 8.6 To find the largest h crit which ensures a monotonic solution (as the analytical one) we execute the following cycle: > > fbvp==inline(' 1 +0. *x', 'x'); for k==3: 1000 [z, u h] == bvp (0, 1 , k, 100, ° , fbv p , ° , 1); if su m (d iff (u h) > °) == == Ie n gt h (u h) -1, break, end, end We let h(= 1/ (k+1)) vary till the forward incremental ratios of the numerical solution uh are all positive. Then we compute the vector diff (uh) whose com- ponents are 1 if the corresponding incremental ratio is positive, 0 otherwise. If the sum of all components equals the vector length of uh diminished by 1, then all incremental ratios are positive. The cycle stops when k=49, that is, when h == 1/500 if 8 == 1000, and when h == 1/1000 if 8 == 2000. We can therefore guess that one should require h < 2/8 == h crit in order to get a monotonically increasing numerical solution.

Indeed, this restriction on h is precisely what can be proven theoretically (see, for instance, [QV94]). In Figure 9.14 we show the numerical solutions obtained when 8 == 100 for two values of h.

9.8 Chapter 8 243

.

: , : , 0.8 ......,.. 0..... ... 0...'..0.....0' .:..0...,'.....,..:... 0..0.0....., o.

....o.....,. 0" - : , : : , : : , - - , 0.6 ,.... 0'...'..... .....,'.
o...,..,.:.....,.,.....,',.

....., "...........

....,. 0..... .,.. : : : , : : - , . : : , : : :, 0.4 ,.....,."..............,.....,.:.....,
0.....' 0 , .

....., ... , . .

.... , . , ..." .. . : :' : :, : :' : ., 0.2 ,.....,.....,.....,..,.....' ':.............."..

..... 0..... 0' '

....,.,...,..... . : - - I : : : I - : _: I _....: ," - I)

..,'" ,: I ,: I ", ;

o

-0.2 o

: ....- ;

;

0.2

0.4

0.6

0.8

Fig. 9.14. Numerical solution for Problem 8.6 obtained for h == 1/10 (dashed line) and h == 1/60 (continuous line)

Solution 8.7 We should modify the Program 17 in order to impose Neumann boundary conditions. In the Program 21 we show one possible implementation.

function [x, uh]==neumann{ a, b, N ,delta ,gamma ,bvpfun, ua ,ub, varargin) h == (b-a)/(N+1); x == [a:h:b]; e == ones(N+2,1); A == spdiags([-e-O.5*h*delta 2*e+gamma*h'"2 -e+O.5*h*delta], -1:1, N+2, N+2); f == h'"2*feval{bvpfun,'x',varargin{:}); f==f'; A{1,1)==-3/2*h; A(1,2)==2*h; A(1,3)==-1/2*h; f(1)==h'"2*ua; A(N+2,N+2)==3/2*h; A{N+2,N+1)==-2*h; A(N+2,N)==1/2*h; f(N+2)==h'"2*ub; uh == A\f;

Solution 8.8 The trapezoidal integration formula, used on the two subinter- vals lk-l and lk, produces the following approximation

J

h h f(X)<pk(X) dx

2 f (Xk) + 2 f (Xk) = hf(Xk),

lk-l Ul k

since CPk(Xj) == 8 jk , Vj, k. Thus, we obtain the same right-hand side of the finite difference method.

Solution 8.9 We have \1</J == (8</J/8x, 8</J/8y)T and therefore div\1</J == 82</J/8x2+ 8 2 </J / 8y2, that is, the Laplacian of </J.

Solution 8.10 To compute the temperature at the center of the plate, we solve the corresponding Poisson problem for various values of Llx == Ll y , using the following instructions:

> > k==O; fun==inline('25', 'x' ,'y'); bound==inline(' (x====l)' ,'x' ,'y'); » for N == [10,20,40,80,160], [u , x, y] == poi 550 n fd (0,0, 1, 1 , N , N , fun, bo un d) ; k==k+1; uc(k) == u(N/2+1,N/2+1); end

The components of the vector lie are the values of the computed temperature at the center of the plate as the step-size h of the grid decreases. We have

» uc 2.0168 2.0616 2.0789 2.0859 2.0890

We can therefore conclude that at the center of the plate the temperature is approximatively 2.08 C. In Figure 9.15 we show the level-curve of the temper- ature for two different values of h.

: l

o o

0.2

0.4

0.6 0.8

Fig. 9.15. The level-curve of the computed temperature for

x ==

y == 1/10 (dashed lines) and for

x ==

y == 1/80 (continuous lines)

Bibliography

[Arn73]

Arnold V. (1973) Ordinary Differential Equations. The MIT Press, Cambridge, Massachusetts. Atkinson K. (1989) An Introduction to Numerical Analy- sis. John Wiley, New York. Axelsson O. (1994) Iterative Solution Methods. Cam- bridge University Press, New York. Brassard G. and Bratley P. (1996) Fundamentals of Algo- rithmics, lie. Prentice Hall, New York. Bernardi C. and Maday Y. (1992) Approximations Spec- trales des Problemes aux Limites Elliptiques.

Springer- Verlag, Paris. Braess D. (1997) Finite Elements: Theory, Fast Solvers and Applications in Solid Mechanics. Cambridge Univer- sity Press, Cambridge. Babuska I. and Strouboulis T. (2001) The Finite Element Method and its Reliability. Oxford University Press. Butcher J. (1987) The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods. Wiley, Chichester.

[Atk89]

[Axe94]

[BB96]

[BM92]

[Bra97]

[BS01]

[But87]

[CHQZ88]

Canuto C., Hussaini M. Y., Quarteroni A., and Zang T. A. (1988) Spectral Methods in Fluid Dynamics. Springer, New York.

[Dav63]

Davis P. (1963) Interpolation and Approximation. Blais- dell Pub., New York. Deuflhard P. and Bornemann F. (2002) Scientific Computing with Ordinary Differential Equations. Springer- Verlag, New York.

[DB02]

[DD95]

Davis T. and Duff I. (1995) A Combined Unifrontal/Multifrontal Method for Unsymmetric Sparse Matrices. TR-95-020. University of Florida.

[Die93]

Dierckx P. (1993) Curve and Surface Fitting with Splines. Claredon Press, New York.

[DL92]

DeVore R. and Lucier J. (1992) Wavelets. Acta Numerica pages 1-56. Davis P. and Rabinowitz P. (1975) Methods of Numerical Integration. Academic Press, New York.

[DR75]

[DS83]

Dennis J. and Schnabel R. (1983) Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, Englewood Cliffs, New York.

[dV89]

cler Vorst H. V. (1989) High Performance Preconditioning. SIAM J. Sci. Stat. Comput. 10: 1174-1185.

[EEHJ96]

Eriksson K., Estep D., Hansbo P., and Johnson C. (1996) Computational Differential Equations. Cambridge Univ. Press, Cambridge.

[EKH02]

Etter D., Kuncicky D., and Hull D. (2002) Introduction to MATLAB 6. Prentice Hall.

[Fun92]

Funaro D. (1992) Polynomial Approximation of Differen- tial Equations. Springer-Verlag, Berlin.

[Gau97]

Gautschi W. (1997) Numerical Analysis. An Introduction. Birkhauser, Berlin.

[GL89]

Golub G. and Loan C. V. (1989) Matrix Computations. The John Hopkins Univ. Press, Baltimore and London.

[Hac85]

Hackbush W. (1985) Multigrid Methods and Applications. Springer-Verlag, Berlin.

[Hac94]

Hackbush W. (1994) Iterative Solution of Large Sparse Systems of Equations. Springer-Verlag, New York.

[HHOO]

Higham D. and Higham N. (2000) MATLAB Guide. SIAM, Philadelphia. Higham N. (1996) Accuracy and Stability of Numerical Algorithms. SIAM Publications, Philadelphia, P A. Hirsh C. (1988) Numerical Computation of Internal and External Flows, volume 1. John Wiley and Sons, Chich- ester.

[Hig96]

[Hir88]

[HLRO 1]

Hunt B., Lipsman R., and Rosenberg J. (2001) A guide to MATLAB: for Beginners and Experienced Users. Cam- bridge University Press.

[IK66]

Isaacson E. and Keller H. (1966) Analysis of Numerical Methods. Wiley, New York.

[Iro70]

Irons B. (1970) A Frontal Solution Program for Finite Element Analysis. Int. J. for Numer. Meth. in Engng. 2: 5-32.

[Kro98]

Kroner D. (1998) Finite volume schemes in multidimen- sions. Pitman Res. Notes Math. Ser., 380, Longman, Har- low.

[KS99]

Karniadakis G. and Sherwin S. (1999) Spectral/hp Ele- ment Methods for CFD. Oxford University Press.

[Lam91]

Lambert J. (1991) Numerical Methods for Ordinary Dif- ferential Systems. John Wiley and Sons, Chichester.

[Lan03]

Langtangen H. (2003) Computational Partial Differen- tial Equations: Numerical Methods and Diffpack Program- ming. Springer-Verlag, Heidelberg.

[LeV02]

LeVeque R. (2002) Finite Volume Methods for Hyperbolic Problems. Cambridge University Press, Cambridge.

[Mei67]

Meinardus G. (1967) Approximation of Functions: Theory and Numerical Methods. Springer-Verlag, New York.

[Pan92] Pan V. (1992) Complexity of Computations with Matrices and Polynomials. SIAM Review 34: 225-262.

[PBP02]

[PdDKUK83]

[QSSOO]

[QV94]

[RR85]

[Saa92]

[Saa96]

[SR97]

[TW98]

[Ueb97]

[Urb02]

[Wes91]

[Wil65]

Prautzsch H., Boehm W., and Paluszny M. (2002) Bezier and B-Spline Techniques. Springer-Verlag.

.. Piessens R., de Doncker-Kapenga E., Ueberhuber C., and Kahaner D. (1983) QUADPACK: A Subroutine Package for

Automatic Integration. Springer- Verlag, Berlin and Heidelberg. Quarteroni A., Sacco R., and Saleri F. (2000) Numerical Mathematics, volume 37 of Texts in Applied Mathematics. Springer-Verlag, New York. Quarteroni A. and Valli A. (1994) Numerical Approxima- tion of Partial Differential Equations. Springer, Berlin and Heidelberg. Ralston A. and Rabinowitz P. (1985) A First Course in Numerical Analysis. McGraw-Hill, Singapore. 7h print- . Ing. Saad Y. (1992) Numerical Methods for Large Eigenvalue Problems. Halstead Press, New York. Saad Y. (1996) Iterative Methods for Sparse Linear Sys- tems. PWS Publishing Company, Boston. Shampine L. F. and Reichelt M. W. (1997) The MATLAB ODE Suite. SIAM J. Sci. Comput. 18: 1-22. Tveito A. and Winther R. (1998) Introduction to Par- tial Differential Equations. A Computational Approach. Springer Verlag.

.. Ueberhuber C. (1997) Numerical Computation: Methods, Software, and Analysis. Springer- Verlag, Berlin Heidel- berg. Urban K. (2002) Wavelets in Numerical Simulation. Springer Verlag. Wesseling P. (1991) An Introduction to Multigrid Methods. John Wiley and Sons, Chichester. Wilkinson J. (1965) The Algebraic Eigenvalue Problem. Clarendon Press, Oxford.

40 45 89 91 93 98 111 125 141 . . . . 145 . . 146 157 158 162 177

Index of MAT LAB Programs

bisection newton rnidpointc trapc

bisection method . . . . . Newton's method . . . . . composite midpoint quadrature formula composite trapezoidal and Gauss formulae . composite Simpson quadrature formula adaptive Simpson formula . . . . . Gauss factorization . . . . General iterative method .... power method. . . . . inverse power method with shift . Gershgorin circles. . .. ... forward Euler method. ... backward Euler method . . . Crank-Nicolson method . . . predictor-corrector method ..... one step of forward Euler (eeonestep), one step of backward Euler (eionestep), one step of Crank-Nicolson (cnonestep) . . . . Approximation of a 2-point boundary-value problem by the finite difference method ..

. Slrnpsonc sirnpadpt lu_gauss iterrneth

. elgpower invshift gershcircles feuler beuler cranknic predcor onestep

bvp

178

192

250 Index of MA TLAB Programs

neUlllann

Approximation of the Poisson problem with Dirichlet data by the five-point finite differ- ence method ..... 200 the Heun method . . . . . 235 Explicit Runge- K utta method of order 3 . . 236 approximation of a Neumann boundary prob-

poissonfd

rk2 rk3

lem. 243

Index

. . . , 31 = 28 ,

approximation least square, 76 arpackc, 150 average, 82 axis, 146

A-stability, 167 abs, 6 absolute error, 3 stability, 165 Adams-Bashforth formula, 175 Adams-Moulton formula, 175 adaptive formula, 95 adaptivity, 73, 95 algorithm, 24 Gauss, 108 multigrid, 134

Strassen, 25 Thomas, see Thomas algo- rithm Winograd and Coppersmith, 25 aliasing, 70 angle, 6 ans, 27

backward difference formula, 176 Euler method, 156 finite difference, 85 substitutions, 107 Bairstow method, 52 basis, 3 bisection, 40 bisection method, 38 boundary conditions, 189, 205 boundary-value problem, 187 Dirichlet, 189 Neumann, 189 break, 209 Broyden method, 52

cancellation, 5

Cauchy problem, 155 Chebyshev interpolation, 65, 66 nodes, 65 chol, 113 Cholesky factorization, 113 choline, 134 clear, 28 compass, 7 compatibility relation, 122 complex, 6 complex number, 6 complexity, 25 computational cost, 25 cond, 118 condest, 118 condition number, 118, 199 conj,7 consistency, 202 consistent method, 160, 202 conv, 18 convergence, 23 Euler method, 158 finite differences, 202 Gauss-Seidel method, 127 interpolation, 65 iterative method, 122, 123 Newton method, 43 order, 23 power method, 142 Richardson method, 130 cos, 28 cputime, 26 Crank-Nicolson method, 162 cumtrapz,91

diag, 11 diff, 20 discretization step, 156 disp, 209 divergence operator, 189

eig, 148 eigenvalue, 14, 137 extremal, 140 eigenvector, 14, 137
eigs, 149 elastic membrane, 201 end, 26 eps, 3 error absolute, 3, 23
computational, 23 estimator, 24,44,45,50,95, 119 interpolation, 63
perturbation, 169 relative, 3, 23 roundoff, 3 truncation, 23, 203 etime,
26 eulesp, 157 exit, 27 exp, 28 explicit method, 157 exponent, 3
extrapolation Richardson method, 101

eye, 9

dblquad,99 deconv, 18 deflation, 52, 150 degree of exactness, 89
Dekker- Brent method, 52 det, 111 determinant, 110

factorization Cholesky, 113, 144 Gauss, 109 LV, 106, 144 QR, 152
feval, 15, 33 Fibonacci sequence, 30 find, 40 finite

Index 253

differences method, 190, 195 elements method, 192 finite difference
backward, 85 centered, 85 forward, 85 fix, 209 fixed point, 4 7
iterations, 48 floating-point numbers, 1 operations, 25 format, 2
formula adaptive, 95 backward difference, 176 Euler, 8 forward Euler
method, 156, 165 finite difference, 85 substitutions, 107 Fourier
transform discrete, 68 fast, 69 fplot, 14, 73 function derivative, 20
graph, 14 primitive, 19 fun tool, 21 fzero, 16, 52 zerof, 157

griddata3,80 griddatan,80

heat equation, 188 help, 28 Heun method, 1 77 Hilbert matrix, 115, 132 hold off, 146 hold on, 146

Gauss algorithm, 108 factorization, 109 nodes, 90 quadrature formulae, 90, 99 Gauss-Seidel method, 126 Gershgorin circles, 146 gmres, 134 grid, 15 griddata,80

if, 26 ifft, 69 imag, 7 implicit method, 157 improved Euler method, 1 77 increment, 129 Inf,4 inline,33 int, 20 interpl, 73 interplq, 73 interp2, 79 interp3, 79 interpft, 69 interpolant, 60 interpolation Chebyshev, 65 composite, 72, 79 Lagrangian polynomial, 60 nodes, 59 piecewise linear, 72 polynomial, 60 rational, 60 splines, see spline trigonometric, 60, 67 inv, 10 invshift, 144 iteration function, 48 iterative methods, 122

Jacobi method, 124

Kronecker symbol, 61, 68

Krylov methods, 134

triangular, 12 tridiagonal, 120, 127, 191 Vandermonde, 109 Wilkinson, 151 mesh, 199 contour, 238 meshgrid, 80, 238 method Adams-Bashforth, 175 Adams-Moulton, 175 backward Euler, see back- ward Euler method Bairstow, 52 bisection, see bisection method Broyden, 52 conjugate gradient, 132, 134 Crank-Nicolson, see Crank- Nicolson method Dekker-Brent, 52 dynamic Richardson, 130 finite element, 204 forward Euler, see forward Euler method Gauss-Seidel, 126 GMRES, 134 gradient, 130 Jacobi, 124 least squares, 75 Monte Carlo, 208 Muller, 52 multifrontal, 134 Newton, see Newton method Newton-Horner, 52 power, see power method predictor-corrector, 1 76 QR, 148 relaxation, 127 SOR, 136 spectral, 204 stationary Richardson, 130 Sturm, 52 midpoint formula, 88

Lagrange characteristic polynomials, 62 form, 62 Lanczos method, 150 Laplace operator, 187, 197 rule, 10 least squares method, 75 lexicographic order, 197 Lipschitz continuous function, 155 load, 28 local truncation error, 159, 203 loglog, 23 Lotka and Leslie model, 138 Lotka- Volterra equations, 154 lu, 110 luinc, 134

magic, 134 mantissa, 3 matrix, 8 bidiagonal, 120 companion, 52 determinant, 10 diagonal, 11 diagonally dominant, 112, 124 Hilbert, see Hilbert matrix identity, 9 ill conditioned, 119 inverse, 10 iteration, 123 permutation, 114 positive definite, 112, 127 prod uct, 9 similar, 148 sparse, see sparse matrix square, 9 strictly diagonal, 127 symmetric, 12, 112, 127 transpose, 12

composite, 87 model problem, 165 generalized, 168 Muller method, 52 multistep method, 175

pivoting, 113 by row, 114 complete, 225 Poisson equation, 187 poly, 35 polyder, 19, 65 polyfit,78 polyint, 19 polynomial, 17 characteristic, 137 Lagrangian interpolation, 60 Taylor, 20 polyval, 17,62 power, 141 power method, .140 inverse, 144 with shift, 144 preconditioner, 123, 130 incomplete L U factorization, 134 predictor-corrector method, 1 76 pretty, 208 problem boundary-value, 187 Cauchy, 155 prod, 209 ptomedioc,89

NaN, 5 nargin, 33 nchoosek, 209 Neumann boundary conditions, 205 newton, 45 Newton method, 42, 54 adaptive, 44 modified, 44 Newton-Cotes formulae, 99 Newton-Horner method, 52 norm, 14 numerical integration, 87

odel13,178 ode15s, 176, 183 ode23, 175,185,240 ode23s, 183 ode23tb, 175,183 ode45,175 one-step method, 156 ones, 13 ordinary differential equation, 153 overflow, 4, 5

QR factorization, 152 method, 148 qr, 152 quad8, 94 quadl, 94, 99 quadrature nodes, 94 weights, 94 quadratures interpolatory, 94 quasi-Newton methods, 52 quit, 27

partial derivative, 187 partial differential equation, 153 patch, 146 path, 30 pcg, 134 pchip, 75 pdetool,80, 202, 204 phase plane, 1 79 piecewise linear interpolation, 72 pivot element, 109

rand, 26

Rayleigh quotient, 137 rcond, 118 real, 7 realmax,4 realmin, 4 rectangle formula, 88 composite, 88 . reglon of absolute stability, 184, 185, 234 regression line, 78, 82 residual, 45, 119, 128 return, 32 root, 17 condition, 164 roots, 17,52 roundoff error, 3, 114, 115 rpmak, 80 rsmak, 80 Runge's function, 65, 67 Runge-Kutta method, 174

spline, 73 cubic, 79, 81 natural cube, 74 spy, 199 sqrt, 28 stability absolute, 168 conditioned absolute, 167 region of absolute, 184, 185, 234 unconditioned absolute, 167 stencil, 197 stiff problem, 182 Strassen algorithm, 25 Sturm method, 52 sequences, 150 sum, 209 syms, 21, 227 system linear, 103 nonlinear, 52 triangular, 106 tridiagonal, 120 underdetermined, 108

save, 28 scalar product, 13 scheme five point, 197 shape function, 194 shift, 144 sign, 40 significant digits, 3 simpadpt, 98 simple, 227 Simpson adaptive formula, 96 composite formula, 93 formula, 93 sin, 28 size, 10 sparse, 199 sparse matrix, 121, 199 spdemos,80 spdiags, 120 spectral radius, 123 spectrum, 140

taylor, 20 Taylor polynomial, 20, 58 taylortool, 59 tlleorem equivalence, 164 first mean-value, 19 of integration, 19 Thomas algorithm, 120, 191 trapezoidal formula, 90 composite, 89 trapz, 91 trigonometric interpolant, 67 tril, 12 triu, 12

underflow, 4

Van Der Pol system, 183 vander, 109 varargin, 15 variance, 82, 219 vector, 13 column, 9 component, 13 linearly independent, 13 norm, 14 row, 9

wavelet, 80 wavelets, 80 weak formulation, 193 whos, 10 wilkinson, 151

zero multiple, 16 simple, 16 zero-stability, 163 zeros, 9, 13

wave equation, 188

1\9Jtt03

Editorial Policy

1. Textbooks on topics in the field of computational science and engineering will be considered. They should be written for courses in CSE education. Both graduate and undergraduate textbooks will be

published in TCSE. Multidisciplinary topics and multidisciplinary teams of authors are especially welcome.

2. Format: Only works in English will be considered. They should be submitted in camera - ready form according to Springer-Verlag's specifications. Electronic material can be included if appropriate. Please contact the publisher. Technical instructions and/or TEX macros are available via http://www.springer.de/math/ authors/help - mom u.html.

3. Those considering a book which might be suitable for the series are strongly advised to contact the publisher or the series editors at an early stage. Addresses are given on the next page.

General Remarks

TCSE books are printed by photo-offset from the master-copy delivered in camera- ready form by the authors. For this purpose Springer-Verlag provides technical instructions for the preparation of manuscripts. See also Editorial Policy. Careful preparation of manuscripts will help keep production time short and ensure a satisfactory appearance of the finished book. The following terms and conditions hold: Regarding free copies and royalties, the standard terms for Springer mathematics monographs and textbooks hold. Please write to Peters@springer.de for details. Authors are entitled to purchase further copies of their book and other Springer books for their personal use, at a discount of 33,3 % directly from Springer-Verlag.

Series Editors

Timothy J. Barth NASA Ames Research Center NAS Division Moffett Field, CA 94035, USA e-mail: barth@nas.nasa.gov Michael Griebel Institut ffir Angewandte Mathematik der Universitat Bonn Wegelerstr. 6 D-53115 Bonn, Germany e-mail: griebel@iam.uni-bonn.de David E. Keyes Computer Science Department Old Dominion University Norfolk, VA 23529-0162, USA e-mail: keyes@cs.odu.edu Risto M. Nieminen Laboratory of Physics Helsinki University of Technology 02150 Espoo, Finland e-mail: rni@fyslab.hut.fi

Dirk Roose Department of Computer Science Katholieke Universiteit Leuven Celestijnenlaan 200A 3001 Leuven-Heverlee, Belgium e-mail: dirk.roose@cs.kuleuven.ac.be Tamar Schlick Department of Chemistry Courant Institute of Mathematical Sciences New York University and Howard Hughes Medical Institute 251 Mercer Street New York, NY 10012, USA e-mail: schlick@nyu.edu Editor at Springer: Martin Peters Springer- Verlag, Mathematics Editorial IV Tiergartenstrasse 17 D-69121 Heidelberg, Germany Tel.: *49 (6221) 487-8185 e-mail: peters@springer.de http://www.springer.de/math/peters.html

Texts in Computational Science and Engineering

Vol. 1 H. P. Langtangen, Computational Partial Differential Equations. Numeri- cal Methods and Diffpack Programming. 2nd Edition 2003. XXVI, 855 pp. Hard- cover. ISBN 3-540-43416-X Vol. 2 A. Quarteroni, F. Saleri, Scientific Computing with MATLAB. 2003. IX, 257 pp. Hardcover. ISBN 3-540-44363-0

Lecture Notes in Computational Science and Engineering

Vol. 1 D. Funaro, Spectral Elements for Transport-Dominated Equati()ns. 1997. X, 211 pp. Softcover. ISBN 3-540-62649-2 Vol. 2 H. P. Langtangen, Computational Partial Differential Equations. Numer- ical Methods and Diffpack Programming. 1999. XXIII, 682 pp. Hardcover. ISBN 3-540-65274-4 Vol. 3 W. Hackbusch, G. Wittum (eds.), Multigrid Methods v: Proceedings of the Fifth European Multigrid Conference held in Stuttgart, Germany, October 1-4, 1996. 1998. VIII, 334 pp. Softcover. ISBN 3-540-63133-X Vol.4 P. Deuflhard, J. Hermans, B. Leimkuhler, A. E. Mark, S. Reich, R. D. Skeel (eds.), Computational Molecular Dynamics: Challenges, Methods, Ideas. Proceed- ings of the 2nd International Symposium on Algorithms for Macromolecular Mod- elling, Berlin, May 21-24, 1997. 1998. XI, 489 pp. Softcover. ISBN 3-540-63242-5 Vol. 5 D. Kroner, M. Ohlberger, C. Rohde (eds.), An Introduction to Recent Devel- opments in Theory and Numerics for Conservation Laws. Proceedings of the Inter- national School on Theory and Numerics for Conservation Laws, Freiburg I Litten- weiler, October 20-24, 1997. 1998. VII, 285 pp. Softcover. ISBN 3-540-65081-4 Vol.6 S. Turek, Efficient Solvers for Incompressible Flow Problems. An Algorith- mic and Computational Approach. 1999. XVII, 352 pp, with CD-ROM. Hardcover. ISBN 3-540-65433-X Vol. 7 R. yon Schwerin, Multi Body System 81M ulation. Numerical Methods, Algorithms, and Software. 1999. XX, 338 pp. Softcover. ISBN 3-540-65662-6 Vol.8 H.-J. Bungartz, F. Durst, C.Zenger (eds.), High Performance Scientific and Engineering Computing. Proceedings of the International FORTWIHR Conference on HPSEC, Munich, March 16-18, 1998. 1999. X, 471 pp. Softcover. 3-540-65730-4

Vol.9 T. J. Barth, H. Deconinck (eds.), High-Order Methods for Computational Physics. 1999. VII, 582 pp. Hardcover. 3-540-65893-9 Vol. 10 H. P. Langtangen, A. M. Bruaset, E. Quak (eds.), Advances in Software Tools for Scientific Computing. 2000. X, 357 pp. Softcover. 3-540-6 6 557-9 Vol. 11 B. Cockburn, G. E. Karniadakis,

C.- W. Shu (eds.), Discontinuous Galerkin Methods. Theory, Computation and Applications. 2000. XI, 470PP. Hardcover. 3-540-667 8 7-3 Vol. 12 U. van Rienen, Numerical Methods in Computational Electrodynamics. Linear Systems in Practical Applications. 2000. XIII, 375 pp. Softcover. 3-540-676 2 9-5 Vol. 13 B. Engquist, L. Johnsson, M. Hammill, F. Short (eds.), Simulation and Vi- sualization on the Grid. Parallelldatorcentrum Seventh Annual Conference, Stock- holm, December 1999, Proceedings. 2000. XIII, 301 pp. Softcover. 3-540-67 26 4-8 VOI.14 E. Dick, K. Riemslagh, J. Vierendeels (eds.), Multigrid Methods VI. Proceedings of the Sixth European Multigrid Conference Held in Gent, Belgium, September 27-30, 1999. 2000. IX, 293 pp. Softcover. 3-540-67 1 57-9 Vol. IS A. Frommer, T. Lippert, B. Medeke, K. Schilling (eds.), Numerical Chal- lenges in Lattice Quantum Chromodynamics. Joint Interdisciplinary Workshop of John yon Neumann Institute for Computing, Jiilich and Institute of Applied Com- puter Science, Wuppertal University, August 1999. 2000. VIII, 184 pp. Softcover. 3-540-67732-1 VOI.16 J. Lang, Adaptive Multilevel Solution of Nonlinear Parabolic PDE Systems. Theory, Algorithm, and Applications. 2001. XII, 157 pp. Softcover. 3-540-67900-6 Vol. 17 B. I. Wohlmuth, Discretization Methods and Iterative Solvers Based on Do- main Decomposition. 2001. X, 197 pp. Softcover. 3-540-41083-X VOI.18 U. van Rienen, M. Giinther, D. Hecht (eds.), Scientific Computing in Elec- trical Engineering. Proceedings of the 3rd International Workshop, August 20-23, 2000, Warnemiinde, Germany. 2001. XII, 428 pp. Softcover. 3-540-42173-4 VOI.19 I. Babuska, P. G. Ciarlet, T. Miyoshi (eds.), Mathematical Modeling and Numerical Simulation in Continuum Mechanics. Proceedings of the International Symposium on Mathematical Modeling and Numerical Simulation in Continuum Mechanics, September 29 - October 3, 2000, Yamaguchi, Japan. 2002. VIII, 301 pp. Softcover. 3-540-42399-0 Vol.20 T. J. Barth, T. Chan, R. Haimes (eds.), Multiscale and Multiresolution Meth- ods. Theory and Applications. 2002. X, 389 pp. Softcover. 3-540-42420-2 Vol. 21 M. Breuer, F. Durst, C. Zenger (eds.), High Performance Scientific and En- gineering Computing. Proceedings of the 3rd International FORTWIHR Confer- ence on HPSEC, Erlangen, March 12-14, 2001. 2002. XIII, 408 pp. Softcover.

3-540-42946-8 Vol.22 K. Urban, Wavelets in Numerical Simulation. Problem Adapted Construc- tion and Applications. 2002.

181 pp. Softcover. 3-540-43055-5

Vol. 23 L. F. Pavarino, A. Toselli (eds.), Recent Developments in Domain Decom- position Methods. 2002. XII, 243 pp. Softcover. 3-540-43413-5 Vol.24 T. Schlick, H. H. Gan (eds.), Computational Methods for Macromolecules: Challenges and Applications. Proceedings of the 3rd International Workshop on Algorithms for Macromolecular Modeling, New York, October 12-14, 2000. 2002. IX, 504 pp. Softcover. 3-540-43756-8 Vol. 25 T. J. Barth, H. Deconinck (eds.), Error Estimation and Adaptive Discretiza- tion Methods in Computational Fluid Dynamics. 2003. VII, 344 pp. Hardcover. 3-540-43758-4 Vol.26 M. Griebel, M. A. Schweitzer (eds.), Meshfree Methods for Partial Differ- ential Equations. 2003. IX, 466 pp. Softcover. 3-540-4389 1 - 2 Vol.27 S. Muller, Adaptive Multiscale Schemes for Conservation Laws. 2003. XIV, 181 pp. Softcover. 3-540-44325-8 Vol. 28 C. Carstensen, S. Funken, W. Hackbusch, R. H. W. Hoppe, P. Monk (eds.), Computational Electromagnetics. Proceedings of the GAMM Workshop on "Com- putational Electromagnetics", Kiel, Germany, January 26-28, 2001. 2003. X, 209 pp. Softcover. 3-540-44392-4 Vol. 29 M. A. Schweitzer, A Parallel Multilevel Partition of Unity Method for El- liptic Partial Differential Equations. 2003. V, 194 pp. Softcover. 3-540-00351-7 Vol.30 T. Biegler, o. Ghattas, M. Heinkenschloss, B. van Bloemen Waanders (eds.), Large-Scale PDE-Constrained Optimization. 2003. approx.344 pp. Softcover. 3-540-05045-0 Vol. 31 M. Ainsworth, P. Davies, D. Duncan, P. Martin, B. Rynne (eds.) Topics in Computational Wave Propagation. Direct and Inverse Problems. 2003. VIII, 394 pp. Softcover. 3-540-00744-X

For further information on these books please have a look at our mathematics catalogue at the following URL:

http://www.springer.de/math/ index. html